精通 Matlab 7

[美] Duane Hanselman
Bruce Littlefield 著
朱仁峰 译

清 华 大 学 出 版 社
北 京

内 容 简 介

Matlab 是适合多学科、多种工作平台的功能强大、界面友好且开放性很强的大型优秀应用软件，同时也是国内外高等院校高等数学、数值分析、数字信号处理、自动控制理论以及工程应用等课程的基本教学、实验仿真工具。本书全面阐述了 Matlab 的所有关键特性和功能，提供了精通 Matlab 所需要的所有方法和手段，通过实例向读者展示如何编写高效的 Matlab 代码。

本书是基于 Matlab 7 编写的。与之前的版本相比，Matlab 7 添加和修改了一些内核数值算法，能支持各种数据类型的数学运算，而不仅仅是双精度类型的数组。Matlab 7 的命令解释程序还增加了一个加速特性——Matlab JIT 加速器（Matlab JIT-Accelerator）。对于 Matlab 7 的新功能，本书用专门的章节进行了详细的介绍。

本书体系完整，深入浅出，实例丰富，既可作为理工科院校研究生、本科生系统学习的教材，也可以作为广大科技人员和教师的参考手册。

前　言

本书是一本关于 Matlab 的参考书，适用于正在使用或将要使用 Matlab 的读者。在本书指导下，无论你是否使用 Matlab 软件自带的帮助文档，都可以完成 Matlab 自学。本书近乎口语化的语言，使读者读起来轻松自如。另外，正如书名所言，本书将向读者提供精通 Matlab 所需要的所有方法和手段。作为一种编程语言和数据可视化工具，Matlab 提供了丰富的方法和手段来解决工程、科学、计算和数学等学科中的问题。本书的主要目的是指导读者如何有效运用这些手段和方法提高工作效率。鉴于 Matlab 本身具有的交互性，本书大部分素材都以实例代码的方式给出，读者可以边读本书，边将感兴趣的代码复制到 Matlab 运行环境中进行验证。

本书内容是针对普通读者的需要设置的。书中的素材可以适用于所有的计算机平台。本书没有专门讨论各类工具箱（Toolbox）、模块集（Blockset）以及其他一些读者需要通过额外付费才能得到的库（Library），但将在适当的地方引用其中的部分内容。因为我们知道，一本书是不可能涵盖所有的与 Matlab 相关的内容的。不过，根据用户反馈，本书将讨论 Matlab 与 C、FORTRAN 和 JAVA 语言之间的链接接口问题。另外，本书还向读者阐明如何将 Matlab 动态链接到计算机平台中的其他应用程序中。

Matlab 作为一个软件工具，版本在不断升级，本书将集中讨论 Matlab 7。本书中的大部分素材同样可以用于 Matlab 6.x 和 Matlab 5.x。需要时，我们会给出不同版本之间的区别。

衷心感谢 MathWorks 公司的全体成员，尤其是 Penny Anderson、Rich Ellis、Tim Farajian、Paul Fricker、Steve Lord、Bob Gilmore 和 Peter Webb。他们认真审校了本书，并提出了许多宝贵建议，使本书内容更加完善。

作为本书的作者，我们热切期望您就本书提出宝贵的意见和建议。例如：“本书最好的特色是什么？”，“哪些章节还需要进一步修改？”，“哪些章节应该被删除？”，“还应该增加哪些内容？”等。您可以通过如下的 E-mail 地址与我们取得联系：mm@eece.maine.edu。另外，您可以在如下网址找到本书的勘误表、实例代码以及其他相关资料：http://www.eece.maine.edu/mm。

Duane Hanselman
Bruce Littlefield

EISBN: 0-13-143018-1

图书在版编目（CIP）数据

精通 Matlab 7/（美）亨塞尔曼（Hanselman, D.），
（美）利特菲尔德（Littlefield, B.）著；朱仁峰译.
—北京：清华大学出版社，2006.5
书名原文：Mastering Matlab 7
ISBN 7-302-12947-9

Ⅰ. 精…　Ⅱ. ①亨…②利…③朱…
Ⅲ.计算机辅助计算—软件包，MATLAB 7　Ⅳ. TP391.75

中国版本图书馆 CIP 数据核字（2006）第 041910 号

出 版 者：清华大学出版社　　**地　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　编**：100084
社 总 机：010-62770175　　**客户服务**：010-82896445
组稿编辑：夏非彼
文稿编辑：洪英
封面设计：林陶
版式设计：科海
印 刷 者：北京市耀华印刷有限公司
发 行 者：新华书店总店北京发行所
开　本：185×260　**印张**：41.75　**字数**：1015 千字
版　次：2006 年 5 月第 1 版　2006 年 5 月第 1 次印刷
书　号：ISBN 7-302-12947-9/TP・8227
印　数：0 001~3 500
定　价：69.00 元

目　　录

Chapter 1

开 始 学 习

1.1 简介

本书的读者需要对矩阵和计算机编程有一定的了解。从总体上讲，矩阵和数组是 Matlab 的核心，因为 Matlab 中所有的数据都是用数组表示和存储的。除了常用的矩阵代数运算之外，Matlab 还提供了各种数组运算功能用于对各种数据集合进行处理。虽然 Matlab 是面向矩阵的编程语言，但它具有与其他计算机编程语言（如 C、FORTRAN）类似的编程特性。在进行数据处理的同时，Matlab 还提供了各种图形用户接口（GUI）工具，以方便用户进行各种应用程序开发。总之一句话，Matlab 把数组数据结构、编程特性和图形用户接口工具集成在一起，成为解决各类问题的一个功能强大的工具。本书将详细讨论 Matlab 的上述几大特点。为了方便读者学习，书中还给出了大量详细的实例代码供参考。

1.2 Matlab 7 的新增内容

Matlab 7 是 Matlab 的最新升级版本。Matlab 7 的界面并没有太大改变，命令（Command）窗口仍然是用户主界面，图形（Figure）窗口用来显示图形信息和创建图形用户接口（GUI），文本编辑器用来创建和编辑 Matlab 代码。Matlab 桌面用来调整其他一些窗口的位置和可视性，如工作区（Workspace）窗口、编辑器（Editor）窗口、帮助（Help）窗口、命令行历史记录（Command History）窗口等。

除了以上的不变特性以外，Matlab 7 在一些数值表示和操作方法上有了新的变化。Matlab 7 添加和修改了一些内核数值算法，能支持各种数据类型的数学运算，而不仅仅是双精度类型的数组（这一数据类型曾一度是较早 Matlab 版本的核心）。更重要的一点是，Matlab 7 的命令解释程序增加了一个加速特性，称为 Matlab JIT 加速器（Matlab JIT-Accelerator）。这一概念最早出现在 Matlab 6.5 版本中，它将一个循环视为一个整体进行代码解释和代码执行而非逐行处理（Matlab 6.5 之前的版本就是这样处理的），从而大大提高了循环操作执行的速度。加速特性省去了代码向量化的过程，用户不再需要使用数组和数组运算就能获得最佳运行性能。不过，要使用 JIT 加速器，用户编写的循环操作代码

必须遵循一定的准则（详见本书后面章节）。如果没有遵循这些准则，则循环操作代码将按照传统的方法逐行被解释。

总的来说，Matlab 7 是对 Matlab 版本演进过程中的又一次改进，其基本操作及功能并没有显著变化。几乎所有用 Matlab 6 编写的代码都可以不加修改地在 Matlab 7 中运行。大部分 Matlab 7 新增和改进的特性都是为了使用户在利用 Matlab 解决问题时取得更高的工作效率而添加的。

1.3 本书的内容

Matlab 软件自身带有大量的文档资料，有打印和电子两个版本，包括了超过 5000 页的产品信息和帮助文本。本书的目的不是为用户提供一个包罗万象的指南或参考手册，也无法将所有的 Matlab 函数囊括其中。本书的主要目的包括：①向初学者介绍 Matlab。②阐明 Matlab 的所有关键特性和功能。③通过实例向读者展示如何编写高效的 Matlab 代码。

如果您不喜欢参考 Matlab 帮助文档，本书无疑是一本极具价值的参考资料。本书的编写遵循这样的原则：用尽可能少的篇幅涵盖尽可能多的信息。即使您喜欢参考 Matlab 帮助文档，本书也是必不可少的工具，因为它提供了大量行之有效的 Matlab 实例代码，充分展示了 Matlab 的诸多特性是如何被用来解决实际问题的。

限于篇幅，有些 Matlab 内容本书没有涉及，如没有详细讨论 Matlab 用户接口中所有的窗口、菜单、菜单项、子菜单、对话框等。当然，这并不意味着这些内容不重要，只是从学习效果来看，对这些内容的书面形式表述还不如别人的言传身教来得快。因此，本文内容将集中在有助于解决实际问题的数学运算、程序编写和图形显示等特性的讲解上。

本书是基于 Matlab 7.0 编写的。随着 Matlab 从 7.0 版本向 8.0 版本发展，书中的有些内容必然会发生变化。这样一来，Matlab 的一些最新特性必然无法在书中体现出来，甚至有些地方给出的信息会变成过时或错误的信息。我们无法控制 Matlab 的发展，也不能仅仅为了 Matlab 新版本中的细微改动而重新编写本书。不过，Matlab 发行商在引入新特性和改变旧特性时是非常谨慎认真的，大部分被改变的旧特性通常都能够在 Matlab 的主干版本中继承下来，有时候还会在一些分支版本中继承下来。因此，尽管本书是针对 Matlab 7.0 编写的，但它同样适用于几乎所有的 7.x 版本。

最后，我们建立了一个名为 Matering MATLAB 的网站来支持本书的学习，网址为：http://www.eece.maine.edu/mm。在该网站中，您可以找到本书的勘误表，以及书中所有用方框括起来的 Matlab（.m）文件。我们热切期望读者通过如下的 E-mail 地址对本书提出宝贵的反馈意见和建议：mm@eece.maine.edu。

Chapter 2

基本特性

运行 Matlab 后，用户的计算机显示器上将弹出一个或多个窗口。其中有一个标题为 Matlab 的窗口是 Matlab 的主用户界面，称为 Matlab 桌面。Matlab 桌面中有一个标题为 Command Window 的窗口，是 Matlab 与用户的主交互区，称为命令窗口。命令窗口中会显示一个提示符“>>”，并且当该窗口处于激活状态时，提示符的右侧会显示一个闪动的光标，这表明 Matlab 正等待用户输入指令，以便执行一项数学运算或其他操作。

2.1 简单的数学运算

用户可以像使用计算器一样，使用 Matlab 进行基本的数学运算。我们来看一个简单的例子：Mary 去办公用品商店买了 4 块橡皮，每块 25 美分；6 本记事簿，每本 52 美分；2 盘磁带，每盘 99 美分，那么，Mary 究竟买了多少件办公用品？这些办公用品总共花了多少钱呢？

如果使用计算器来解决上述问题，您可能会做如下输入：

```
4+6+2=12（项）
4×25+6×52+2×99=610（美分）
```

在 Matlab 中，上述问题可以用多种方法解决。首先，可以按照与使用计算器相同的方法直接在 Matlab 提示符后输入，如下所示[①]：

```
>> 4+6+2
ans =
    12
>> 4*25+6*52+2*99
ans =
    610
```

注意：在大多数情况下，输入行中的空格不会对 Matlab 运算产生影响。另外，在 Matlab

① 译者注：为了使读者阅读方便，我们在程序中保留了“>>”提示符，以表明这些命令可以在命令窗口中输入验证。另外，对于计算的结果，我们也尽量保持了 Matlab 原始的显示方式。

中，乘法的优先级高于加法。还有，在前面两次运算中，由于没有指定输出结果的名称，Matlab 将默认的运算结果命名为 ans，这是单词 answer 的简写。

其次，上述问题还可以通过将信息存储在 Matlab 变量中来解决，如下所示：

```
>> erasers = 4
erasers =
     4
>> pads = 6
pads =
     6
>> tape = 2;
>> items = erasers + pads + tape
items =
     12
>> cost = erasers*25 + pads*52 + tape*99
cost =
     610
```

这里，我们生成了 3 个 Matlab 变量：erasers、pads 和 tape，分别存储每种办公用品的数量。在上述各语句中，除输入 tape 的那条语句以外，每条语句输入之后，Matlab 就会立即显示出计算结果。Tape 语句行之所以没有显示输出结果，是因为该行后面的分号告诉 Matlab 只对该行求解，不用显示结果。最后，我们将购买的办公用品总数命名为 items，将总金额命名为 cost，并利用数学公式求出了它们的值。

注意：Matlab 在每一步都会记住先前运算的信息，这是因为 Matlab 具有记忆功能。我们不妨来计算一下平均每件办公用品的价格是多少，如下所示：

```
>> average_cost = cost/items
average_cost =
    50.833
```

因为 average cost 是两个单词，而 Matlab 变量名必须是一个单词，因此我们用了一个下划线来将这两个词连接成一个 Matlab 变量 average_cost。

Matlab 提供的基本数学运算如下表所示：

运算	符号	示例
加法	+	3 + 22
减法	-	54.4 - 16.5
乘法	*	3.14 * 6
除法	/ 或 \	19.54 / 7 或 7 \ 19.54
乘方	^	2 ^ 8

在一个给定的表达式中，上述运算的优先级与我们常用的优先级规则是一样的，这个规则可以概括如下：

> *表达式将按从左到右的顺序进行运算，其中指数运算的优先级最高；乘法和除法次之，二者具有相同的优先级；加法和减法的优先级最低，二者也具有相同的优*

先级。圆括号将改变上述优先级顺序，但上述优先级在同一圆括号内仍旧适用，表达式具有多重圆括号时，其优先级从外到内依次升高。

读者可以在命令窗口中输入 help precedence 查看更多关于优先级顺序的信息。

2.2 Matlab 工作区

上节提到，Matlab 具有记忆功能。当用户在命令窗口中操作时，Matlab 会记录下用户所输入的命令和创建的所有变量的值。这些命令和变量都被保存在一个标题为 Workspace 的 Matlab 工作区或基本工作区窗口中，并且可以在用户需要的任何时候调用。例如，如果我们要查看一下 tape 的值，只需在提示符后输入 tape 即可，如下所示：

```
>> tape
tape =
     2
```

如果用户记不起某个变量的名称，可以使用 Matlab 命令 who 让 Matlab 列出当前的变量列表，如下所示：

```
>> who
Your variables are:
ans                 cost             items        tape
average_cost        erasers          pads
```

需要注意的是，这里 Matlab 并没有显示变量的值，而仅仅列出了它们的变量名。要想获得它们的值，用户必须在 Matlab 提示符后输入相应的变量名。

用户可以使用键盘上的方向键重新调用原来输入过的命令。例如，按一次↑键，就可以重新调用当前操作命令之前的那条操作命令。重复按下↑键，就可以依次调用更靠前的那些操作命令，直到第一条操作命令。类似地，按一次↓键，就可以重新调用当前操作命令之后的那条操作命令。按下→键和←键就会使光标在当前命令行中左右移动，这样，用户在编辑这些命令时就像在字处理软件中编辑文本一样轻松自如。其他的一些标准编辑按键，例如 Delete 键、Backspace 键、Home 键和 End 键都执行它们通常所执行的操作。Tab 键常用于变量名的拼写。当用户用↑键或↓键选定了命令历史记录中的某条命令，不管此时光标处在这条命令的什么位置，只要按下回车键（Return），Matlab 就开始执行这条命令。最后，Esc 键可以清除当前的命令（该键用户可能用得比较少，但比较有用）。为了满足那些熟悉了 EMACS 编辑器的用户，Matlab 也支持常用的 EMACS 编辑中的组合控制键，例如，可以用 Control-U 来清除当前的命令。

2.3 关于变量

与其他计算机编程语言一样，Matlab 也有自己的一套变量命名规则。前文中我们已经提到，变量名必须是一个单一的词，不能包含空格。详细的 Matlab 变量命名规则如下：

变量命名规则	注释或示例
变量名区分大小写	Cost、cost、CoSt 和 COST 都是不同的变量名
变量名最多能包含 63 个字符，其后的字符都被忽略	Howaboutthisvariablename
变量名必须以一个字母开始，其后可以是任意数量的字母、数字或者下划线	how_about_this_variable_name X51483
不允许出现标点符号，因为很多标点符号在 Matlab 中有特殊的意义	a_b_c_d_e

当然，除了上述规则之外，还有一些特殊规定。例如，Matlab 中的关键字（又称为保留字）不能用作 Matlab 变量名。Matlab 的关键字列表如下：

保留字列表
`For end if while function return elseif case otherwise`
`switch continue else try catch global persistent break`

Matlab 函数 iskeyword 的返回值就是上述列表中的内容。如果用户把这些关键字用作变量名，Matlab 将会发出一条错误信息。但是，如果将这些关键字中的某个或者某些字母改成大写，用户就可以用与这些关键字类似的词作为变量名。用户可以用函数 isvarname('teststring')验证字符串“teststring”是否为合法的 Matlab 变量名：若是则函数返回 True（即 1），否则返回 False（即 0）。

另外，Matlab 还定义了如下一些特殊变量：

特殊变量	描述
ans	用作结果的默认变量名
beep	使得计算机发出“嘟嘟”声
pi	圆周率
eps	浮点精度限（2.2204×10^{16}），Matlab 中的最小数，如该数与 1 相加，将产生大于 1 的最小的那个数
inf	表示无穷大，例如 1/0
NaN 或 nan	表示不定数，即结果不能确定，例如 0/0
i 或 j	虚数，表示 $\sqrt{-1}$
nargin	函数的输入参数个数
nargout	函数的输出参数个数
realmin	可用的最小正实数值
realmax	可用的最大正实数值
bitmax	可用的最大正整数（以双精度格式存储）
varargin	可变的函数输入参数个数
varargout	可变的函数输出参数个数

如果用户再次使用一个用过的变量（如 2.1 节例子中的 tape 变量），或者给上表中的特殊变量重新赋值，则它们原来的值就会被覆盖。不过，用这个变量原来的值计算的其他表达式的值不会受到影响。下面是具体的例子：

```
>> erasers = 4;
>> pads = 6;
>> tape =2;
>> items = erasers + pads + tape
items =
    12
>> erasers = 6
erasers =
     6
>> items
items =
    12
```

这里，我们仍然使用 2.1 节的例子。首先得出 Mary 购买的办公用品总数。然后，将橡皮的数量设为 6，覆盖其原来的值 4。读者可以发现 items 的值并没有改变。与我们常用的电子制表程序不同，Matlab 不会根据 erasers 这个变量值的改变实时刷新购买办公用品的总数。Matlab 执行一项运算时，将根据运算请求发出时它所知道的变量的值来完成运算。上例中，如果用户希望重新计算办公用品总数、总的金额和平均价格，就必须重新调用相应的 Matlab 命令，让 Matlab 重新计算。

特殊变量也遵循上述规则，但是，特殊变量的值可以被自动恢复。也就是说，当用户重新启动 Matlab 时，特殊变量的值就恢复到它的初始值；一旦用户改变了它们的值，其初始值便丢失了。另外，用户要使特殊变量恢复到初始值而不重新启动 Matlab，只需执行 clear 命令将新值覆盖即可，如下所示：

```
>> pi
ans =
    3.1416
>> pi = 1.23e-4
pi =
    0.000123
>> clear pi
>> pi
ans =
    3.1416
```

上例表明，pi 有 5 个有效数字，其初始值为 3.1416，我们用 1.23e-4 覆盖了其初始值，然后，用 clear 函数清除了这个新值，pi 又恢复到初始值。

2.4　注释、标点符号和中止执行

我们先前已经看到，在一条命令的末尾输入一个分号，Matlab 就不会在屏幕上显示这条命令计算的结果。当我们不希望显示计算的中间结果时，这一特性尤其有用。例如：

```
>> erasers
erasers =
     6
>> items = erasers + pads + tape;
```

```
>> cost = erasers*25 + pads*52 + tape*99;
>> average_cost = cost/items
average_cost =
     47.143
```

上例将 Mary 购买的橡皮数量由原来的 4 个改为 6 个，并最终计算这些办公用品的平均价格。其中 items 和 cost 为中间结果，我们利用分号使其计算结果不在屏幕上显示。

除了分号之外，Matlab 还使用其他标点符号。比如，在一个百分号（%）之后的所有文本都被看作是一条注释，如：

```
>> tape = 2 % number of rolls of tape purchased
```

变量 tape 被赋值为 2，百分号和百分号之后的文本都将被 Matlab 忽略。

此外，我们还可以利用逗号或分号在一行中输入多条命令，例如：

```
>> erasers = 6, pads = 6; tape = 2
erasers =
    6
tape =
    2
```

其中以逗号结尾的表达式要显示结果，而以分号结尾的表达式不显示结果。

有时候，一个表达式或命令很长，这时我们就不得不另起一行书写。在 Matlab 中，可以用 3 个连续的句点（...，称为续行符）表示同一语句的延续输入，例如：

```
>> average_cost = cost/items % command as done earlier
average_cost =
      47.143
>> average_cost = cost/... % command with valid continuation
items
average_cost =
      47.143
>> average_cost = cost... % command with valid continuation
/items
average_cost =
      47.143
>> average_cost = cost...  command with valid continuation (no % needed)
/items
average_cost =
      47.143
>> average_cost = cost/it... % command with INvalid continuation
ems
??? ems
    |
Error: Missing MATLAB operator.
```

注意：只有当续行符出现在变量名和数学运算符之间时，才能起到语句延续的作用，当出现在一个变量名的中间位置时，是不能实现语句延续的（这时必然会报错）。换句话说，一个变量名不能被隔开分散在两行。另外，由于注释行是被忽略的（*所有百分号后的内容*都不起作用），因此注释行不能利用续行符实现续行，例如：

```
>> % Comments cannot be continued...
>> either
??? Undefined function or variable 'either'.
```

上例中，注释行末尾的续行符只能作为注释的一部分，Matlab 不会处理这个续行符。最后，可以在任何时候利用组合键 Control-C 来中止 Matlab 的执行过程。

2.5 复数

Matlab 最强大的一个特性就是它无需做任何特殊操作，就可以对复数进行处理。在 Matlab 中，复数的产生方式很多，下面的示例代码给出了其中的几种典型方式：

```
>> c1 = 1-2i % the appended i signifies the imaginary part
c1 =
    1.0000-2.0000i
>> c1 = 1-2j % j also works
c1 =
    1.0000-2.0000i
>> c1 = complex(1,2) % a function that creates complex numbers
c1 =
    1.0000-2.0000i
>> c2=3*(2-sqrt(-1)*3)
c2 =
    6.0000-9.0000i
>> c3 = sqrt(-2)
c3 =
    0+1.4142i
>> c4 = 6+sin(.5)*1i
c4 =
    6.0000 + 0.4794i
>> c5 = 6+sin(.5)*1j
c5 =
    6.0000 + 0.4794i
```

在生成 c4 和 c5 时，我们分别通过乘以 1i 和 1j 来获得虚部。这一操作是必需的，因为 sin(.5)i 和 sin(.5)j 在 Matlab 中是没有任何意义的。因此，我们可以发现，只有数字才能与字符 i 和 j 直接连接，而表达式则不可以。

在有些编程语言中，如果出现复数，就需要进行特殊处理，而 Matlab 则不需要。在 Matlab 中，复数数学运算的写法与实数数学运算的表达方式相同，如下所示：

```
>> c6 = (c1+c2)/c3 % from the above data
c6 =
     -7.7782-4.9497i
>> c6r = real(c6)
c6r =
     -7.7782
>> c6i = imag(c6)
c6i =
     -4.9497
```

```
>> check_it_out = 1i^2 % sqrt(-1) squared must be -1!
check_it_out =
    -1
```

一般而言，复数运算的结果也是复数。不过，如果所得结果的虚部为 0 时（如上例最后一种情况），Matlab 会自动去掉结果中的 0 虚部。另外，上例中分别使用了函数 real 和 imag 来获取一个复数的实部和虚部。

最后，我们再给出一个复数运算的例子。我们都知道欧拉恒等式，这个恒等式将一个复数的极坐标形式和它的直角坐标形式联系起来，即：$M\angle\theta = Me^{j\theta} = a + bj$，其中极坐标表达式用极径 M 和角度 θ 表示，直角坐标形式用 $a + bj$ 表示。这两种表达式之间的关系是 $M=\sqrt{a^2+b^2}$，$\theta=\tan^{-1}(b/a)$，$a=M\cos(\theta)$，$b=M\sin(\theta)$。

在 Matlab 中，我们可以使用函数 real、imag、abs 和 angle 完成极坐标和直角坐标表达式之间的转换，如下面的代码所示：

```
>> c1
c1 =
    1.0000-2.0000i
>> mag_c1 = abs(c1) % magnitude
mag_c1 =
    2.2361
>> angle_c1 = angle(c1) % angle in radians
angle_c1 =
    -1.1071
>> deg_c1 = angle_c1*180/pi % angle in degrees
deg_c1 =
    -63.4349
>> real_c1 = real(c1) % real part
real_c1 =
    1
>> imag_c1 = imag(c1) % imaginary part
imag_c1 =
    -2
```

在上面的代码中，Matlab 函数 abs 用来计算一个复数的模或一个实数的绝对值，Matlab 函数 angle 用来计算一个复数的角度（单位为弧度）。

注意：在以前的版本中，Matlab 无法以度数为单位执行三角运算。不过，在 Matlab 7 中，所有的基本三角函数都能够支持以度数为单位的角度值作为输入参数。

2.6 浮点运算

几乎在所有情况下，Matlab 中的数值都是用双精度数来表示的，这些双精度数在系统内部用二进制表示。二进制是计算机最常用的表示方式，同时也是数字协处理器的固有格式。然而，正是由于这种表示方式，使得并非所有的数值都能被准确地表示，也就是说，Matlab 中的数值表示存在一个极限值，另外，在进行加法运算时，还存在一个公认的下限。

Matlab 存在一个它能够表示的最大正实数，我们可以用下面的代码获得：

```
>> format long % tell MATLAB to display more precision
>> realmax
ans =
    1.797693134862316e+308
```

同样，Matlab 也存在一个它能够表示的最小正实数，我们可以用下面的代码获得：

```
>> realmin
ans =
    2.225073858507201e-308
```

另外，Matlab还存在一个用双精度值表示的浮点相对误差限eps，定义为1与比它大的最小数之间的距离，即该值加上1所产生的数是比1大的数中最小的数。我们可以用下面的代码获得eps：

```
>> eps
ans =
    2.220446049250313e-016
```

推而广之，eps(x)将产生x与比它大的最小数之间的步进距离。如下面的代码所示：

```
>> eps(1) % same as eps by itself
ans =
    2.220446049250313e-016

>> eps(10)
ans =
    1.776356839400251e-015

>> eps(1e10)
ans =
    1.907348632812500e-006
```

由上面的代码可见，随着 x 数量级的增加，由有限精度表示的数值之间的距离（即上述 eps(x)的返回值）也会随之增大。

Matlab 中有限精度的局限往往会产生非常奇怪的结果。例如，下边这个例子表明加法并不是绝对满足交换律：

```
>> 0.42 - 0.5 + 0.08
ans =
    -1.387778780781446e-017

>> 0.08 - 0.5 + 0.42 % rearrange order
ans =
    0

>> 0.08 + 0.42 - 0.5 % rearrange order again
ans =
    0
```

从数学角度讲，上边的 3 个表达式的计算结果都应该是 0，但实际上并非如此。出现这一问题的原因在于：并不是所有的数字都能够用双精度数精确地表示。实际上，在上面的 3 个数中，只有 0.5 可以被精确表示。当数字不能被精确表示时，Matlab 就会给它们一

个尽可能精确的近似值——这就给计算的结果带来了不可避免的误差。在多数情况下，这些误差是很小的，否则现代计算机中就不可能再使用双精度数了。实际上，双精度数带来的误差通常只在用 Matlab 比较两个数是否相等时才会出现。因此，根据上面的分析，尽管我们知道 0.42-0.5+0.08 与 0.08-0.5+0.42 是完全相等的，但在 Matlab 中，二者却不相等。

有限精度效应所带来的第二个后果出现在函数运算中。一方面 Matlab 不能精确地表示函数的参数，另一方面，大多数函数本身也无法被精确地表示。我们看下面的代码：

```
>> sin(0)
ans =
     0
>> sin(pi)
ans =
     1.224646799147353e-016
```

根据数学知识，上述两个结果都应该是 0，但 sin(π)的结果却明显不为 0。另外，细心的读者可以发现，本例和前边那个例子中出现的误差都小于 eps。

最后，Matlab 中的整数也是用双精度浮点数来表示的。Matlab 中所有的整数都可以被精确地表示，但存在它所能表示的最大整数上限，我们可以用下面的代码获取这一上限：

```
>> bitmax
ans =
     9.007199254740991e+015
```

这个值等于 $2^{53}-1$。

2.7 数学函数

Matlab 提供了一系列的函数来支持基本的数学运算。其中大多数函数的用法和我们平时书写数学表达式时的用法一样，如下面的代码所示：

```
>> x = sqrt(2)/2
x =
    0.7071
>> y = asin(x)
y =
    0.7854
>> y_deg = y*180/pi
y_deg =
    45.0000
```

上述代码用来求正弦值为 $\sqrt{2}/2$ 时的角度值。再次提醒读者注意：在进行三角函数运算时，Matlab 使用的是弧度值，而不是角度值。Maltab 提供的数学函数很多，下面给出了其他一些数学函数的用法示例：

```
>> y = sqrt(3^2 + 4^2) % show 3-4-5 right triangle relationship
y =
    5
>> y = rem(23,4) % remainder function, 23/4 has a remainder of 3
```

```
y =
     3
>> x = 2.6, y1 = fix(x), y2 = floor(x), y3 = ceil(x), y4 = round(x)
x =
     2.6000
y1 =
     2
y2 =
     2
y3 =
     3
y4 =
     3
```

关于其他一些数学函数，我们不再一一举例，下表给出了基本数学函数的函数名和简单描述：

三角函数	描述
acos	反余弦函数
acosd	反余弦函数，返回度数值
acosh	反双曲余弦函数
acot	反余切函数
acotd	反余切函数，返回度数值
acoth	反双曲余切函数
acsc	反余割函数
acscd	反余割函数，返回度数值
acsch	反双曲余割函数
asec	反正割函数
asecd	反正割函数，返回度数值
asech	反双曲正割函数
asin	反正弦函数
asind	反正弦函数，返回度数值
asinh	反双曲正弦函数
atan	反正切函数
atand	反正切函数，返回度数值
atanh	反双曲正切函数
atan2	四个象限内反正切
cos	余弦函数
cosd	余弦函数，以度数为输入参数
cosh	双曲余弦函数
cot	余切函数
cotd	余切函数，以度数为输入参数

（续表）

三角函数	描述
coth	双曲余切函数
csc	余割函数
cscd	余割函数，以度数为输入参数
csch	双曲余割函数
sec	正割函数
secd	正割函数，以度数为输入参数
sech	双曲正割函数
sin	正弦函数
sind	正弦函数，以度数为输入参数
sinh	双曲正弦函数
tan	正切函数
tand	正切函数，以度数为输入参数
tanh	双曲正切函数

指数函数	描述
^	求乘方
exp	求指数
expm1	指数减 1（即 exp(x)-1）
log	求自然对数
log10	求以 10 为底的对数
log1p	求 x+1 的自然对数
log2	求以 2 为底的对数，用于浮点数分割
nthroot	求实数的第 n 个实根
pow2	求以 2 为底的幂，用于浮点数换算
reallog	求非负实数的自然对数
realpow	求非负实数的乘方
realsqrt	求非负实数的平方根
sqrt	求平方根
nextpow2	最小的 p，使得 2^p 不小于给定的数 n

复数函数	描述
abs	求实数的绝对值或复数的模
angle	求以弧度为单位的相角
conj	求复数的共轭值
imag	求复数的虚部

（续表）

复数函数	描述
real	求复数的实部
unwrap	复数的相角展开
isreal	判断是否为实数，若是，返回 True，否则返回 False
cplxpair	将矢量按共轭复数对重新排序
complex	由实部和虚部创建复数

取整和求余函数	描述
fix	向 0 取整
floor	向负无穷取整
ceil	向正无穷取整
round	向最近的整数取整
mod	求模或有符号取余
rem	求除法的余数
sign	符号函数

坐标变换函数	描述
cart2sph	笛卡尔坐标到球坐标变换
cartpol	笛卡尔坐标到柱坐标或极坐标变换
pol2cart	柱坐标或极坐标到笛卡儿坐标变换
sph2cart	球坐标到笛卡儿坐标变换

数理函数	描述
factor	求一个数的质数因子
isprime	判断一个数是否为质数，若是，返回 True，否则返回 False
primes	产生不大于某数的质数序列
gcd	求一个数的最大公约数
lcm	求一个数的最小公倍数
rat	有理逼近
rats	有理数输出
perms	求几个元素所有可能的组合
nchoosek	求从 n 个元素中一次取 k 个元素的所有可能的组合

特殊函数	描述
airy	Airy 函数
besselj	第一类贝塞尔函数
bessely	第二类贝塞尔函数
besselh	第三类贝塞尔函数
besseli	经过修正的第一类贝塞尔函数
besselk	经过修正的第二类贝塞尔函数
beta	beta 函数
betainc	不完全 beta 函数
betaln	beta 函数的对数
ellipj	Jacobi 椭圆函数
ellipke	完全椭圆积分
erf	误差函数
erfc	互补误差函数
erfcx	经过比例缩放的互补误差函数
erfinv	误差函数的逆函数
expint	指数误差函数
gamma	gamma 函数
gammainc	不完全 gamma 函数
gammaln	gamma 函数的对数
legendre	Legendre 伴随函数
cross	向量叉乘
dot	向量点乘

Chapter 3

Matlab 桌面

上一章我们提到，运行 Matlab 将会在用户显示器上生成一个或多个窗口。其中有一个标题为 Matlab 的窗口即为 Matlab 桌面。该窗口是管理 Matlab 其他窗口的主窗口。根据用户对 Matlab 的设置不同，Matlab 的有些窗口可见，有些则不可见，有些可以驻留（即嵌入）在 Matlab 窗口内部，有些则不可以。Matlab 的窗口管理与其他基于窗口的应用程序（如微软 Word 2000）大同小异，本书中不再赘述。如果读者对其他基于窗口的应用程序比较熟悉，那么对 Matlab 窗口的操作也不会有太大问题。Matlab 桌面中的菜单项将根据当前哪个窗口处于激活状态而发生变化。另外，Matlab 中有许多有用的上下文菜单（在一个菜单项上单击鼠标右键所显示的菜单）。如果读者对基于窗口的应用程序不太熟悉，可以利用这些上下文菜单从其他资源中获取广泛的帮助信息。如果用户想要知道 Matlab 桌面到底管理哪些窗口，只要浏览一下 Matlab 桌面中的 Desktop 菜单栏即可。

3.1 Matlab 的窗口

Matlab 中常用到的一些窗口包括：命令窗口（Command Window）、命令历史窗口（Command History Window）、当前目录浏览器窗口（Current Directory Browser）、工作区浏览器窗口（Workspace Browser）、帮助浏览器窗口（Help Browser）、编辑器窗口（Editor Window）、剖析器窗口（Profiler Window）。下表列出了上述窗口的功能和用途：

窗口	描述
Command	用于输入命令使 Matlab 进行某项处理
Command History	用于显示或运行在命令窗口中发出过的命令的历史记录
Current Directory	Matlab 中路径和文件管理的图形用户界面（GUI）
Workspace	查看、编辑、装载和保存 Matlab 变量的图形用户界面（GUI）
Help	查找并查看在线帮助文档的图形用户界面（GUI）
Editor	创建 M 文件的文本编辑器
Profiler	用于分析 M 文件运行性能的工具

3.2 管理 Matlab 工作区

在 Matlab 桌面中，所有窗口中执行的操作都支持命令窗口中的运算。因此，我们首先着重介绍一下命令窗口。

在命令窗口中创建的数据和变量都保存在 Matlab 工作区（基本工作区）中。用户除了可以在工作区窗口中查看变量之外，还可以利用 who 命令在命令窗口中查看 Matlab 工作区中的变量，如下所示：

```
>> who
Your variables are:
angle_c1        c4              cost            pads
ans             c5              deg_c1          real_c1
average_cost    c6              erasers         tape
c1              c6i             imag_c1
c2              c6r             items
c3              check_it_out    mag_c1
```

说明：如果读者在计算机上验证，所看到的变量可能和上述变量列表不一样，这取决于在启动 Matlab 之后，让它执行了什么操作。要想得到各个变量更详细的信息，可以用 whos 命令，如下所示：

```
>> whos
  Name            Size       Bytes     Class
  angle_c1        1×1           8      double array
  ans             1×1           8      double array
  average_cost    1×1           8      double array
  c1              1×1          16      double array(complex)
  c2              1×1          16      double array(complex)
  c3              1×1          16      double array(complex)
  c4              1×1          16      double array(complex)
  c5              1×1          16      double array(complex)
  c6              1×1          16      double array(complex)
  c6i             1×1           8      double array
  c6r             1×1           8      double array
  check_it_out    1×1           8      double array
  cost            1×1           8      double array
  deg_c1          1×1           8      double array
  erasers         1×1           8      double array
  imag_c1         1×1           8      double array
  items           1×1           8      double array
  mag_c1          1×1           8      double array
  pads            1×1           8      double array
  real_c1         1×1           8      double array
  tape            1×1           8      double array
Grand total is 21 elements using 216 bytes
```

上面的结果列出了每个变量的大小、字节数及数据类型。由于 Matlab 是*面向数组的*，因此所有的变量都属于双精度数组类型，无论该变量是标量还是向量。在后面的章节中，

随着其他数据类型（或类）被逐渐引入，上述结果的最后一列将会提供更多更有用的信息。另外，上述变量列表也显示在工作区窗口中，在 Matlab 提示符后输入 workspace，或选择 Matlab 桌面中 Desktop 菜单下的 Workspace 菜单项，都可以显示工作区窗口。

如果要清除 Matlab 工作区中的变量，可以使用 clear 命令，如下所示：

```
>> clear real_c1 imag_c1 c*
>> who
Your variables are:
angle_c1          deg_c1          mag_c1
ans               erasers         pads
average_cost      items           tape
```

上面的命令将变量 real_c1、imag_c1 和所有以字母 c 开头的变量从工作区删除。我们可以使用 help 或 helpwin 命令显示 clear 函数的用法信息，如下面的代码所示：

```
>> help clear
CLEAR Clear variables and functions from memory.
   CLEAR removes all variables from the workspace.
   CLEAR VARIABLES does the same thing.
   CLEAR GLOBAL removes all global variables.
   CLEAR FUNCTIONS removes all compiled M- and MEX-functions.

   CLEAR ALL removes all variables, globals, functions, and MEX links.
   CLEAR ALL at the command prompt also removes the Java packages import
   list.

   CLEAR IMPORT removes the Java packages import list at the command
   prompt. It cannot be used in a function.

   CLEAR CLASSES is the same as CLEAR ALL except that class definitions are
   also cleared. If any objects exist outside the workspace(say in userdata
   or persistent in a locked m-file) a warning will be issued and the class
   definition will not be cleared. CLEAR CLASSES must be used if the number
   or names of fields in a class are changed.

   CLEAR JAVA is the same as CLEAR ALL except that java classes on the
   dynamic java path (defined using JAVACLASSPATH) are also cleared.

   CLEAR VAR1 VAR2 ... clears the variables specified. The wildcard
   character'*' can be used to clear variables that match a pattern.For
   instance, CLEAR X* clears all the variables in the current workspace that
   start with X.

   CLEAR -REGEXP PAT1 PAT2 can be used to match all patterns using regular
   expressions. This option only clears variables. For more information on
   using regular expressions, type "doc regexp" at the command prompt.

   If X is global, CLEAR X removes X from the current workspace, but leaves
   it accessible to any functions declaring it global.
   CLEAR GLOBAL X completely removes the global variable X.
   CLEAR GLOBAL -REGEXP PAT removes global variables that match regular
   expression patterns.
   Note that to clear specific global variables, the GLOBAL option must
```

```
come first. Otherwise, all global variables will be cleared.

CLEAR FUN clears the function specified. If FUN has been locked by MLOCK
it will remain in memory. Use a partial path (see PARTIALPATH) to
distinguish between different overloaded versions of FUN. For instance,
'clear inline/display' clears only the INLINE method for DISPLAY,leaving
any other implementations in memory.

CLEAR ALL, CLEAR FUN, or CLEAR FUNCTIONS also have the side effect of
removing debugging breakpoints and reinitializing persistent variables
since the breakpoints for a function and persistent variables are cleared
whenever the m-file changes or is cleared.

Use the functional form of CLEAR, such as CLEAR('name'), when the variable
name or function name is stored in a string.

Examples for pattern matching:
    clear a*                % Clear variables starting with "a"
    clear -regexp ^b\d{3}$  % Clear variables starting with "b" and
                            %    followed by 3 digits
    clear -regexp \d        % Clear variables containing any digits

See also who, whos, mlock, munlock, persistent.
```

很显然，clear 命令不仅仅具有删除变量的功能，随着用户对 Matlab 的各种特性逐渐熟悉，会对其功能有更深入的了解。

3.3 内存管理

当用户创建一个变量或运行一个 M 文件函数时，Matlab 就会为这些变量和函数分配相应的内存空间。根据用户计算机配置的不同，Matlab 有可能会出现内存溢出现象，使得用户无法从事进一步的工作。当用户用 clear 命令删除变量时，Matlab 就释放这个变量所占用的内存。然而，这样多次操作以后，就有可能使得内存碎片化，也就是说，这时 Matlab 的内存空间充斥着由大量碎小闲置内存包围的许多变量。由于 Matlab 总是在内存的连续区域保存变量，因此这些内存碎片对 Matlab 而言不可再用。为了缓解这个问题，pack 命令就用来完成内存碎片收集的工作。该命令先将 Matlab 工作区中所有的变量保存到硬盘上，然后清空工作区，最后再将原有变量重新载入到工作区中。这项操作完成之后，所有的内存碎片就被合并成一个大的、可用的内存块。根据用户的机器有多少内存可以分配给 Matlab、某个 Matlab 程序已经运行了多长时间，以及用户创建了多少个变量，用户可以选择是否使用 pack 命令。

3.4 数字显示格式

Matlab 在显示数值结果时是有章可循的。在默认情况下，如果结果是整数，Matlab 就将结果显示成整数。类似地，如果结果是实数，Matlab 就显示成一个具有 4 位小数位的实数。如果结果的有效位超出了这个范围，Matlab 就用科学计数法显示，这与科学计算器有些相似。当然，用户也可以设置不同的数字格式而不使用默认格式。这一操作很简单，用

户只要在命令窗口的 File 菜单中选择 Preference 菜单项，或者在提示符后输入相应的 Matlab 格式命令即可。以特殊变量 pi 为例，下表列出了不同格式命令下所产生的不同的数字显示格式：

Matlab 格式命令	pi	注释
format short	3.1416	5 位
format long	3.14159265358979	16 位
format short e	3.1416e+000	5 位+指数
format long e	3.14159265358979e+000	16 位+指数
format short g	3.1416	短紧缩格式
format long g	3.14159265358979	长紧缩格式
format hex	400921fb54442d18	16 进制，浮点
format bank	3.14	2 位小数
format +	+	正（+）、负（-）或者 0（0）
format rat	355/113	有理数近似
format debug	Structure address =1214830 m=1 n=1 pr=11d60d0 pi=0 3.1416	短紧缩格式的内部存储信息

> **注意**：选择不同的显示格式并不会改变 Matlab 数值的内部存储方式，只改变数值的显示方式。所有的计算仍旧是用双精度数进行的。

3.5　保留会话日志

有时将 Matlab 一次会话中执行的所有操作保存在日志中往往是非常有用的。在 Command History 窗口中按照时间顺序保留着当前和过去的 Matlab 会话中所执行的函数和命令的日志，但是并不显示结果。Matlab 提供了一个命令 diary，它将命令窗口中所有的操作都存储在当前目录下的一个文本文件中。利用 help 命令可以查看对 diary 命令的描述：

```
>> help diary
DIARY Save text of MATLAB session.
   DIARY filename causes a copy of all subsequent command window input
   and most of the resulting command window output to be appended to
   the named file. If no file is specified, the file 'diary' is used.

   DIARY OFF suspends it.
   DIARY ON turns it back on.
   DIARY, by itself, toggles the diary state.
```

3.6 系统信息

Matlab 提供一些命令用于提供用户当前使用的计算机的信息以及 Matlab 的版本信息。命令 computer 返回一个字符串，标识当前所使用的计算机的信息：

```
>> computer
ans =
PCWIN
```

在这里，当前计算机是一台正在运行 Windows XP 的 PC 机。

命令 version 同样返回一个字符串，标识当前使用的 Matlab 版本：

```
>> version
ans =
7.0.0.151483 (R14)
```

（注意：上述结果会根据用户所安装的 Matlab 版本不同而不同。）

还有一个类似的命令 ver，返回当前 Matlab 的版本信息以及所安装的工具箱：

```
>> ver
---------------------------------------------------------------------
MATLAB Version 7.0.0.051483 (R14)
MATLAB License Number:9754
Operating System: Microsoft Windows XP Version 5.1(Build 2600: Service
Pack 1)
Java VM Version: Java 1.4.2 with Sun Microsystems Inc. Java HostSpot(TM)
Client VM
---------------------------------------------------------------------
MATLAB                                Version 7.0        (R14)
Mastering MATLAB Toolbox              Version 6.0
```

（注意：上述结果会根据用户所安装的 Matlab 版本和操作系统的不同而不同。）

用命令 license 和 hostid 可以查看 Matlab 的许可证信息，例如：

```
>> hosted
   '9754'
>> license
ans =
9754
```

当然，当你在自己的计算机上输入这些命令的时候，所得到的结果可能会和上述显示结果不一致，这是因为你的计算机和 Matlab 版本可能与产生上述结果的计算机和 Matlab 版本不一致。

3.7 Matlab 搜索路径

Matlab 用 search path 来获取存储在硬盘上的文件信息。Matlab 文件都存储在硬盘上的众多目录和子目录下。

> Matlab 文件所在的所有目录的列表被称作 Matlab 搜索路径或者简称为 Matlab 路径。

下面，介绍 Matlab 搜索路径的用法。当你在 Matlab 提示符后输入 cow 之后，Matlab 就完成如下操作：①检查 cow 是不是 Matlab 工作区中的变量名，如果不是，执行下一步。②检查 cow 是不是一个内置函数，如果不是，执行下一步。③检查当前目录下是否存在一个名为 cow.m 的文件，如果没有，执行下一步。④按顺序检查在所有 Matlab 搜索路径中是否存在 cow.m 文件。⑤如果到目前为止还没有找到这个 cow，Matlab 就给出一条错误信息。

Matlab 在执行相应的指令时都是基于上述的搜索策略完成的。上例中，如果 cow 是一个变量，Matlab 就使用这个变量。如果 cow 是一个内置函数，Matlab 就调用这个函数。如果 cow.m 是当前目录或 Matlab 搜索路径中的一个文件，Matlab 就打开这个文件，然后执行这个文件中的指令内容。在第 4 章和第 12 章将会提到，Matlab 有两种基本的文件类型，这两种类型的文件都是包含 Matlab 命令的简单文本文件。（关于 M 文件的更多信息请参看第 4 章和第 12 章。）

实际上，由于 Matlab 高级特性的存在，Matlab 的搜索过程比上面所描述的要复杂得多。但是，在大部分情况下，上述搜索过程对于大多数 Matlab 操作已经足够了。（关于 Matlab 搜索路径的更详细信息，请参看第 12 章。）

当 Matlab 启动的时候，将默认的 Matlab 搜索路径定义为存储 Matlab 应用程序文件的所有目录（即 Matlab 安装路径及其各个子目录）。我们可以通过不同的方式显示和修改这个搜索路径。最简单的方式就是用路径浏览器，它是一个用来查看和修改 Matlab 搜索路径的图形用户界面。选择 Matlab 桌面窗口中 File 菜单下的 Set Path 菜单项，就会弹出这个路径浏览器的窗口。既然 Matlab 的搜索路径默认指向了存储 Matlab 应用程序文件的所有目录，那么访问 Matlab 的路径浏览器的主要目的是为了将用户自己定义的文件目录加入到搜索路径中。

为了在命令窗口中显示 Matlab 搜索路径，Matlab 提供了一个函数 matlabpath。另外，路径浏览器的特性也可以在命令窗口中使用函数 path、addpath 和 rmpath 实现。关于这些函数的更详细信息，请参看 Matlab 的联机帮助文档。

Chapter 4

M 脚本文件

对于一些简单的问题，当所需命令数很少时，快捷而有效的方法是在 Matlab 的命令窗口中直接输入这些命令。但是，对于有些问题，当命令数较多，或者用户需要改变其中的一个或多个变量的值进行重复验证时，直接输入命令的方法就不那么方便了。针对这些问题，Matlab 提供了一个合理的解决方法。该方法允许用户将一系列 Matlab 命令输入到一个简单的文本文件中，只要在 Matlab 命令窗口中打开这个文件，文件中所有的命令都被依次执行，其结果和用户在命令窗口中逐条输入命令完全一样。这样的文本文件在 Matlab 中称为脚本文件，其中"脚本"一词意味着 Matlab 仅仅从这个文件中读取脚本语句；又由于这些文件都以".m"作为文件后缀名（例如，本章中用到的 M 脚本文件名为 example1.m），因此也称为 M 文件。

4.1 M 脚本文件的用法

创建 M 脚本文件的方法有两种：①单击 Matlab 桌面窗口工具栏上的空白页图标（即新建图标）。②选择 File 菜单下的 New 菜单项的 M-file 子菜单项。这些动作完成后，就会弹出一个文本编辑窗口，用户可以在其中输入 Matlab 命令。例如，用户可以将第 2 章提到的"玛丽购物"的例子写成下述 M 脚本文件形式：

```
% script M-file example1.m
erasers = 4; % number of each item
pads = 6;
tape = 2;
items = erasers + pads + tape
cost = erasers*25 + pads*52 + tape*99
average_cost = cost/items
```

编辑完成后，用户可以通过下面 3 种方法将这个文件保存在硬盘上并立即执行：①从 Debug 菜单中选择 Save and Run 菜单项。②单击编辑器工具栏中的 Save and Run 按钮。③按 F5 功能键。如果不想立即执行，用户可以通过选择 File 菜单中的 Save 菜单项，将文件保存为"example1.m"（当然，也可以保存为其他文件名）；若要执行这个文件时，只需

在 Matlab 提示符后输入该 M 文件的文件名（不带后缀.m）即可。下面的代码给出了执行结果：

```
>> example1
items =
    12
cost =
    610
average_cost =
        50.833
```

按照第 3 章讲到的路径搜索优先级原则，Matlab 在接收到 example1 这条语句的时候，首先检查 example1 是不是当前的 Matlab 工作区中存在的变量名和 Matlab 的内置函数名，若都不是，再检查其有效搜索路径下是否存在一个 M 脚本文件名为 example1.m，如果存在，Matlab 就打开这个文件，然后执行文件中的命令。M 文件被打开后，文件中的命令就可以访问当前 Matlab 工作区中的所有变量，并且这个 M 文件所创建的新变量也将被增加到 Matlab 工作区中。通常情况下，Matlab 在执行 M 文件时，文件中的命令并不显示出来。如果用户需要显示，则可用 echo on 命令指示 Matlab 在执行 M 文件中的命令时将这些命令显示在命令窗口中。当不需要显示时，利用 echo off 即可取消显示（本节最后给出了这两个命令的具体应用）。另外，反复执行单个命令 echo，可以实现命令显示状态的来回转换（相当于交替执行 echo on 和 echo off）。

有时用户在解决某类问题时，需要改变其中的一个或多个变量的值进行重复验证（通常称这类问题为 what if 问题），这时利用 M 脚本文件就可以使问题的解决变得简单易行。比如，用户可以反复打开 example1.m 文件，改变 erasers、pads 或者 tape 的值，然后保存并运行这个文件，就可以得到不同的结果。

从 example1.m 中可以看到，使用脚本文件进行 Matlab 运算时，注释是非常必要的。注释是指用户对脚本文件中的命令添加的说明性文字，便于用户日后阅读和理解。另外，在一条命令语句后添加分号意味着在执行该语句时不显示执行结果，这一特点可以用来控制是否显示脚本文件的输出，以便只在屏幕上显示重要的结果。

在 Matlab 脚本文件中，有一些函数对于控制文件执行十分有用，下表给出了这些函数的名称和功能描述：

函数	描述
beep	让计算机发出“嘟嘟”声
disp(variablename)	只显示结果，而不显示变量名
echo	在脚本文件被执行时，控制脚本文件内容是否在 Command 窗口中显示
input	提示用户输入数据
keyboard	临时终止 M 文件的执行，让键盘获得控制权。按回车（Return）键就将控制权交还给正在执行的 M 脚本文件
pause 或 pause(n)	暂停，直到用户按下任何一个键盘按键为止，或者暂停 n 秒后继续执行
waitforbuttonpress	暂停，直到用户按下鼠标键或者键盘按键为止

当一条 Matlab 命令不是以分号结尾时，那么该命令的执行结果连同变量名同时显示在 Command 窗口中。当我们不需要显示变量名，而只需要显示结果时，可以用 disp 函数实现，如下例所示：

```
>> items
items =
    12
>> disp(items)
    12
```

当用户需要针对不同情况反复修改脚本并进行运算时，在脚本中使用 input 函数就可以用来在执行脚本文件的时候提示用户输入变量的值，从而免去了对脚本文件的反复编辑过程。例如，我们可以对 example1.m 脚本文件再进行如下的修改：

```
% script M-file example1.m
erasers = 4; % Number of each item
pads = 6;
tape = input('Enter the number of rolls of tape purchased>');
items = erasers + pads + tape
cost = erasers*25 + pads*52 + tape*99
average_cost = cost/items
```

运行上面的脚本文件，产生结果如下：

```
>> example1
Enter the number of rolls of tape purchased > 3
items =
    13
cost =
    709
average_cost =
        54.538
```

当用户输入 example1 运行后，首先会出现一行提示，要求用户输入磁带的盘数，用户输入数字 3 并回车后，便得到上述结果。

除了一个常数外，input 函数还接受任何有效的 Matlab 表达式作为输入参数。例如，我们再次运行 example1.m 脚本文件，并在输入提示后输入不同的表达式，运算结果如下：

```
>> example1
Enter the number of rolls of tape purchased > round(sqrt(13))-1
items =
    13
cost =
    709
average_cost =
        54.538
```

在上面的例子中，磁带盘数被设置为下述表达式计算的结果（其值仍为 3）：

```
round(sqrt(13))-1.
```

如果在脚本文件中加入 echo 函数，表示要控制命令行的回显效果。例如，我们可以把 echo on 和 echo off 添加到文件中将 example1.m 文件修改如下：

```
% script M-file example1.m
echo on
erasers = 4; % Number of each item
pads = 6;
tape = input('Enter the number of rolls of tape purchased>');
items = erasers + pads + tape
cost = erasers*25 + pads*52 + tape*99
average_cost = cost/items
echo off
```

运行该文件，结果如下：

```
>> example1
erasers = 4; % Number of each item
pads = 6;
tape = input('Enter the number of rolls of tape purchased >');
Enter the number of rolls of tape purchased > 2
Items = erasers + pads + tape
items =
    12
cost = erasers*25 + pads*52 + tape*99
cost =
    610
average_cost = cost/items
average_cost =
        50.833
echo off
```

从运行结果我们可以看到，回显太多的命令行会降低运行结果的可读性，使用户读起来不是那么一目了然。但在调试比较复杂的脚本文件时，echo 命令还是非常有用的。

4.2　块注释和代码单元

在 Matlab 7.0 以前的版本中，注释是逐行进行的。也就是说，若一行为注释行，则该行以一个百分号（%）开始，注释内容直到该行结尾都有效。要想在下一行继续注释内容，必须重新以百分号开始。因此，要想进行块注释（即连续数行注释），每一行都必须以百分号开始，如下例所示：

```
% This is an example of multiple line comments
% in an M-file. Each line requires an initial % sign or MATLAB
% assumes the line contains code to be executed.
```

尽管上述的块注释看起来比较简单，但不利于用户增加和修改注释内容。因为用户每增加一行注释都必须在行首添加一个百分号。在过去的 Matlab 版本中，为了解决这一问题，Matlab 编辑器提供了将一块（若干行）选中的文字转化为注释的工具和命令。在 Matlab 7 中，用户可以使用“%{”和“%}”符号进行块注释。其中，“%{”和“%}”分别代表注释块的起始和结束。比如，上面的注释内容可以使用块注释语法重新表示如下：

```
%{
This is an example of multiple line comments
in an M-file. Each line requires an initial % sign or MATLAB
assumes the line contains code to be executed.
(Now lines can be added and edited as desired without having to
place percent signs at the beginning of each line.)
%}
```

从上面的内容可以看出，块注释可以使用户方便快捷地书写多行注释。另外，在创建和调试大型 M 文件时，块注释还可以帮助用户控制 M 文件中任意一行或多行程序的解释与执行。用户只要在一块文字的首尾添加“%{”和“%}”，就可以使其由代码段变为注释部分，在运行时 Matlab 就不再执行这些代码。应用这一特性，在编辑和调试过程中，用户可以使脚本文件中的不同代码段在不同的时间执行。

在以往的版本中，Matlab 通过编辑器提供的操作命令和工具执行一段选中的代码。在 Matlab 7 中，用户可以使用代码单元完成这一操作。一个代码单元指用户在 M 文件中指定的一段代码，它以一个代码单元符号（双百分号加空格，即“%%”）为开始标志，到另一个代码单元符号结束，如果不存在第二个代码单元符号，则直到该文件结束。Matlab 编辑器中的 Cell 菜单提供了用户创建、单独执行和顺序执行代码单元的命令。注意：代码单元只能在 Matlab 编辑器窗口中创建和使用，在 Matlab 命令窗口中是无效的。也就是说，如果用户在 Matlab 命令窗口中输入含有代码单元的 M 文件并运行时，文件中的代码单元语法是被忽略的，Matlab 将执行所有的代码（包括代码单元中的语句）。

4.3 设置执行时间

在正常情况下，当用户在 Matlab 命令窗口中输入一个 M 文件的文件名并回车后，该文件是被立即执行的。但在有些情况下，例如大型工程项目或长时间执行才能得到所需结果时，用户并不希望文件立即执行，而希望能够控制 M 文件的执行时间。在 Matlab 中，这一功能可以通过使用计时器对象实现。计时器对象可以使用 timer 函数创建，下例创建了一个名为 my_timer 的计时器对象变量：

```
>> my_timer = timer('TimerFcn','MfileName','StartDelay',100)
```

上例计时器对象表明，当用户使用 start 函数启动计时器 100 秒后，名为 MfileName 的 M 文件将被执行。启动计时器的命令如下：

```
>> start(my_ timer) % start the timer in the variable my_timer
```

timer 函数的一般语法定义为：

```
>> t = timer('PropertyName1',PropertyValue1,'PropertyName2',
             PropertyValue2,...)
```

该定义中，参数总是成对出现，分别表示属性的名称和属性值。

上例中，当计时器启动后，Matlab 并不立即执行代码，用户可以利用这一空闲时间从事其他的操作。100 秒后，计时器对象开始代码执行，这时它将代替 Matlab 对代码进行控制，当代码执行结束后，计时器释放对代码的控制权，Matlab 重新将控制权交给命令窗口。

计时器对象还有许多其他特性。首先，'MfileName'可以是任何可执行语句，例如，一个 M 脚本文件、一个函数句柄、一个 M 函数文件或一系列 Matlab 命令等。另外，用户可以利用计时器对象使指定的代码按周期循环执行或一次执行数个周期。最后，一个计时器中可以同时对 4 个 M 文件或代码序列进行不同的定时操作。例如，我们可以创建如下的计时器对象：

```
>> my_timer = timer ('TimerFcn','Mfile1',...
                     'StartFcn','Mfile2',...
                     'StopFcn','Mfile3',...
                     'ErrorFcn','Mfile4');
```

该计时器对象将执行如下操作：①将'Mfile1'作为基本计时器代码循环周期执行。②当使用 start 函数启动计时器时执行'Mfile2'。③当使用 stop 函数终止计时器时执行'Mfile3'。④当 Matlab 出错时执行'Mfile4'。

更多的关于计时器对象的信息请参考相应的 Matlab 帮助文档。

4.4 启动和终止

当 Matlab 启动时，将运行两个 M 脚本文件 matlabrc.m 和 startup.m。第一个文件 matlabrc.m 主要用于设置图形显示（Figure）窗口的默认大小和位置，以及一些其他默认特性，该文件是和 Matlab 一起被安装在硬盘上的，通常不能进行修改。另外，matlabrc.m 文件还用于设置默认的 Matlab 搜索路径，这一工作是通过调用脚本文件 pathdef.m 来完成的。当用户用路径浏览器和一些命令函数重新设置搜索路径时，都无形中更改了文件 pathdef.m 的内容，因此用户没有必要专门编辑和修改这个文件。

当 matlabrc.m 文件运行时，通常要检测 Matlab 搜索路径中是否存在 startup.m 脚本文件，如果存在，就执行文件中的命令。startup.m 是一个可选文件，通常包含用户添加的一些默认特性。比如，用户可以在 startup.m 文件中添加一个或多个 addpath 或 path 命令，将更多的目录添加到 Matlab 的搜索路径中；另外，还可以用诸如 format compact 这样的命令来改变默认的数字显示格式。因为 startup.m 是一个标准的 M 脚本文件，所以几乎所有的命令都可以添加到该文件中（但建议用户不要将 quit 命令添加到该文件中！）。在单用户安装版本中，startup.m 文件通常保存在 Matlab 安装目录下的 toolbox/local 子目录中；在网络安装版本中，为了调用方便，startup.m 文件通常在用户启动 Matlab 所在的默认目录中。

用户结束 Matlab 运行可以采用两种方法：①在 Matlab 桌面窗口中，选择 File 菜单下的 Exit Matlab 菜单项。②在 Matlab 命令窗口中，输入 exit 或者 quit 命令。当用户采用任何一种方法结束 Matlab 运行时，Matlab 将在其搜索路径中寻找一个名为 finish.m 的文件。如果找到该文件，Matlab 结束运行之前将执行这个文件中的命令。例如，下面这个 finish.m 文件提示用户正试图退出应用程序，并让用户再次确认是否确实要退出，其中的 quit cancel 命令将取消退出操作。

```
% FINISH Confirm Desire for Quitting MATLAB
question = 'Are You Sure You Want To Quit?';
button = questdlg(question, 'Exit Request', 'Yes', 'No', 'No');

switch button
case 'No'
    quit cancel; % how to cancel quitting!
end
% 'Yes' lets script and MATLAB end.
```

Chapter 5

数组和数组运算

到目前为止，我们考虑的所有运算都只涉及到了单个数字，我们称之为标量。标量运算是数学的基础。但是，当用户希望同时对多个数据执行相同运算时，重复的标量运算则显得既耗时又麻烦。为了解决这个问题，Matlab 定义了基于数据数组的运算。

5.1 简单数组

现在，我们来考虑计算正弦函数在半个周期内的取值问题，也就是说，y=sin(x)，0≤x≤π。很显然，我们无法计算这个区间内所有点上的 sin(x)的值（因为在这个区间内有无限多个取值点），因此，我们必须选择有限个点进行计算，即我们对这个函数进行采样。为了选择取值点，我们每隔 0.1π 计算一次 sin(x)的值，也就是说，令 x=0，0.1π，0.2π，…，1.0π。如果用户使用科学计算器来计算这些值的话，需要首先生成一个 x 值的数组或列表；然后将 x 的每个值输入到计算器中，得到相应的正弦值；之后将结果写到另一个数组 y 中；最终用户可能会生成如下格式的一个列表：

x	0	0.1π	0.2π	0.3π	0.4π	0.5π	0.6π	0.7π	0.8π	0.9π	π
y	0	0.31	0.59	0.81	0.95	1	0.95	0.81	0.59	0.31	0

如上表所示，x 和 y 是一一对应的有序数列，也就是说，y 中的第一个值（或元素）与 x 中的第一个值（或元素）是相关联的，y 中的第二个值与 x 中的第二个值是相关联的，依此类推。因为这种有序性的存在，我们很自然想到用下标来表示 x 和 y 中的一个单独元素；例如，x_1 表示 x 的第一个元素，y_5 表示 y 的第 5 个元素，x_n 表示 x 的第 n 个元素。

Matlab 处理数组的方式简单而明了。创建数组更是轻而易举——我们只需遵循前面给出的排列规则就可以创建如下数组：

```
>> x = [ 0 .1*pi .2*pi  .3*pi  .4*pi  .5*pi  .6*pi  .7*pi  .8*pi  .9*pi  pi ]
x =
  Columns 1 through 7
           0   0.3142  0.6283  0.9425  1.2566  1.5708  1.8850
  Columns 8 through 11
      2.1991  2.5133  2.8274  3.1416
>> y = sin(x)
```

```
y =
  Columns 1 through 7
           0   0.3090  0.5878  0.8090  0.9511  1.0000  0.9511
  Columns 8 through 11
      0.8090  0.5878  0.3090  0.0000
```

要想在 Matlab 中创建一个数组，用户只需先输入一个左方括号，然后输入每个数值并用空格（或者逗号）隔开，最后用一个右方括号结束数组创建。在这里，我们可以注意到，x 的正弦值与 x 的值在排序上是自然对应的。也就是说，Matlab 知道用户想要得到 x 中的每个元素的正弦值，并把计算结果按照相应的顺序存储到数组 y 中。这一基本功能是 Matlab 与其他计算机编程语言重要的差别。

上例中我们看到，空格起到分隔数组元素的功能，因此，若数组中含有复数元素，这些元素中不能间杂空格，除非该元素用圆括号括起来。例如，[1 –2i 3 4 5+6i]有 5 个元素，但是由同样的一些数构成的数组[(1 – 2i) 3 4 5+6i]和[1-2i 3 4 5+6i]却只有 4 个元素。

5.2 数组寻址或者下标

在前一节的例子中，x 包含有多个元素（它有 11 个不同的值分布在 11 个不同的列上），Matlab 也在 y 中相应的列上给出了对应的计算结果。从前一节的显示结果我们看到，x 是一个 1 行 11 列的数组，用数学术语描述就是，它是一个行向量、一个 1×11 的数组或者简称为一个长度为 11 的数组。

在 Matlab 中，我们可以通过下标来访问单个数组元素。例如，x(1)是 x 的第一个元素，x(2)是 x 的第二个元素，依此类推。例如：

```
>> x(3) % The third element of x
ans =
    0.6283
>> y(5) % The fifth element of y
ans =
    0.9511
```

为了同时访问一块数据，Matlab 提供了冒号：

```
>> x(1:5)
ans =
        0   0.3142  0.6283  0.9425  1.2566
```

这样我们就得到了 x 数组中第 1 到第 5 个元素。括号中的“1:5”的意思是从 1 开始，然后加 1 计数直到 5。再看下例：

```
>> x(7:end)
ans =
        1.885       2.1991      2.5133      2.8274      3.1416
```

这条命令返回从 x 数组的第 7 个元素到最后一个元素的值。这里，关键字 end 表示 x 数组的最后一个元素。在引用数组元素时，我们可以控制递增顺序和步进值。例如：

```
>> y(3:-1:1)
ans =
    0.5878      0.3090      0
```

这条语句按照逆序返回 y 的第 3 个、第 2 个和第 1 个元素的值。表示法“3:-1:1”表示从 3 开始，向下减 1 计数，到 1 结束。

```
>> x(2:2:7)
ans =
    0.3142      0.9425      1.5708
```

这条语句给出了 x 数组的第 2 个、第 4 个和第 6 个元素的值。表示法“2:2:7”表示从 2 开始，然后以步长为 2 计数，到 7 结束。在这个例子中，2+6=8，大于 7，因此第 8 个元素没有包含在内。我们还可以随机抽取数组中一个或多个元素的值，如下例：

```
>> y([8   2   9   1])
ans =
    0.8090      0.3090      0.5878      0
```

在这里，我们用到了另外一个数组[8　2　9　1]，并按照我们希望的顺序提取数组 y 中的元素。提取的第 1 个元素是 y 中的第 8 个值，第 2 个元素是 y 中的第 2 个值，第 3 个元素是 y 中的第 9 个值，第 4 个元素是 y 中的第 1 个值。实际上，[8　2　9　1]本身就是一个数组，它的作用是指定抽取地址。我们注意下面的例子：

```
>> y([1    1    3    4    2    2])
ans =
         0       0    0.5878  0.8090  0.3090  0.3090
```

在上例中，并没有要求抽取的索引地址必须互不相等。这使得用户可以随意地重新排列和复制数组元素。该特性使 Matlab 编程更具高效性。

Matlab 也允许用其他数组对一个数组进行寻址，只要这个寻址数组的元素都是介于 1 到被寻址数组长度之间的整数。下面的例子都是不合法的：

```
>> y(3.2)
??? Subscript indices must either be real positive integers of logicals.
>> y(11.6)
??? Subscript indices must either be real positive integers of logicals.
>> y(12)
??? Index exceeds matrix dimensions.
```

在上述例子中，当 Matlab 接受到非整数下标值时，将仅仅给出一条报警信息；当 Matlab 接受到一个超过变量长度的整数值时，将返回一条信息提示索引值溢出。当然，在上面任何一种情况下，Matlab 都不会给出数字结果。

5.3　数组结构

在前边，我们通过逐个输入 x 的每个元素的值来给定 x 的值。这种方法在 x 只有 11 个元素的时候还行之有效，但如果 x 有 111 个元素呢？下面我们给出两种输入 x 的值的方法：

```
>> x = (0:0.1:1)*pi
x =
    Columns 1 through 7
              0   0.3142   0.6283   0.9425   1.2566   1.5708   1.8850
    Columns 8 through 11
        2.1991   2.5133   2.8274   3.1416
>> x = linspace(0,pi,11)
x =
    Columns 1 through 7
              0   0.3142   0.6283   0.9425   1.2566   1.5708   1.8850
    Columns 8 through 11
        2.1991   2.5133   2.8274   3.1416
```

在上边第一个例子中，冒号表示法(0:0.1:1)将创建一个从 0 开始，步长为 0.1，到 1 结束的数组。将该数组中的每个元素都乘以 π，就得到数组 x。第二个例子则利用函数 linspace 来创建 x，该函数的参数如下所示：

```
linspace(first_value, last_value, number_of_values)
```

上述两种数组生成方式在 Matlab 中都是很常用的。冒号表示法使用户能够直接指定数据点之间的增量，而不用指定数据点的个数；linspace 函数法则使用户能够直接指定数据点的个数，而不用指定数据点之间的增量。

值得注意的是，上述两种方式所生成的数组其元素是等间隔分布的。但在某些特殊情况下，我们需要创建对数间隔的数组。这一操作可用 Matlab 提供的 logspace 函数实现：

```
>> logspace(0,2,11)
ans =
    Columns 1 through 7
        1.0000    1.5849    2.5119    3.9811    6.3096    10.0000    15.8489
    Columns 8 through 11
        25.1189  39.8107   63.0957   100.0000
```

在这里，我们创建了一个从 10^0 开始，到 10^2 结束，包含 11 个值的数组。函数 logspace 的参数如下所示：

```
logspace(first_exponent, last_exponent, number_of_values)
```

尽管在通常情况下，都是从 10 的整数次方开始，到 10 的整数次方结束，但 logspace 同样可以用非整数作为其前两个参数。

在使用冒号表示法或者函数 linspace 和 logspace 的时候，人们还是习惯于把表达式用方括号括起来，例如：

```
>> a = [1:7]
a =
    1   2   3    4   5   6   7
>> b = [linspace(1,7,5)]
b =
        1         2.5         4         5.5           7
```

虽然添加方括号不会改变计算结果，还使语句变得一目了然，但是添加方括号会使得 Matlab 进行多余的运算，造成时间的浪费，这是因为方括号表示连接操作。在前面的例子

中，我们并不需要进行连接操作，因此就没有必要让 Matlab 花费时间考虑是否进行方括号连接。

与方括号不同，圆括号并不意味着连接操作，因此不会使 Matlab 运行变慢。因此，圆括号可以根据需要任意使用，例如：

```
>> a = (1:7)' % change row to column
a =
     1
     2
     3
     4
     5
     6
     7
```

有时候用户需要这样一个数组，该数组无法方便地用均匀间隔或者对数间隔的元素关系来描述，并且没有一个统一的方法来生成这样的数组。这时，数组的寻址和表达式组合功能使用户不必逐个输入单独元素的值。例如：

```
>> a = 1:5, b =1:2:9
a =
     1     2     3     4     5
b =
     1     3     5     7     9
```

这个语句生成两个数组。请读者注意，如果多个语句用逗号或者分号隔开，则可以在一行输入。

```
>> c = [b a]
c =
     1     3     5     7     9     1     2     3     4     5
```

这条语句生成一个数组 c，数组 c 是用 b 和 a 的元素组成的，其中 b 的元素排在前边，a 的元素排在后边。

```
>> d = [a(1:2:5) 1 0 1]
d =
     1     3     5     1     0     1
```

这条语句生成一个数组 d，d 是由数组 a 的第 1、第 3 和第 5 个元素以及后边的 3 个附加元素构成的。

下表总结了 Matlab 简单的数组创建特性：

数组创建方法	描述
x=[2 2*pi sqrt(2) 2-3j]	创建包含任意元素的行向量 x
x=first:last	创建行向量 x，从 first 开始，步长为 1，到 last 结束，如果不能到 last，则到小于 last 的最大整数结束。要注意，x=[first:last]可以得到相同的数组，但是会花更多的时间，因为 Matlab 会同时考虑方括号的连接操作和冒号的数组创建操作

（续表）

数组创建方法	描述
x=first:increment:last	创建行向量 x，从 first 开始，步长为 increment，到 last 结束，如果不能到 last，则到小于 last 的最大整数结束
x=linspace(first,last,n)	创建均匀间隔的行向量 x，从 first 开始，到 last 结束，总共有 n 个元素
x=logspace(first,last,n)	创建对数间隔的行向量 x，从 10^{first} 开始，到 10^{last} 结束，总共有 n 个元素

5.4 数组方向

在前面所举的例子中，数组都包含一行多列。正因为它们的这种行方向特性，我们通常称之为行向量。不过，数组也有可能是一个一列多行的列向量。这时，前述所有数组操作和数学运算都同样适用而无需做任何改变。惟一的区别就是结果被显示成列而不是行。

既然前述数组生成函数所创建的数组都是行向量，那么也一定存在创建列向量的方法。创建列向量最直接的方法就是逐个指定列向量元素，并用分号隔开，如下例：

```
>> c = [1;2;3;4;5]
c =
     1
     2
     3
     4
     5
```

> 通过这个示例，我们可以得出这样的结论：用空格或者逗号隔开的元素指定了不同列中的元素，而用分号隔开的元素则指定了不同行中的元素。

要想用前面讲到的冒号表示法（start:increment:end）、linspace 函数或 logspace 函数创建列向量，用户必须用 Matlab 转置操作符（'）将它们所生成的行向量转置成列向量。例如：

```
>> a = 1:5
a =
     1     2     3     4     5
```

这条命令用冒号表示法生成一个行向量。

```
>> b = a'
b =
     1
     2
     3
     4
     5
```

这条命令用转置操作符将行向量 a 转置成列向量 b。

```
>> w = b'
w =
     1     2     3     4     5
```

这条语句再次使用转置操作符，将列向量变回行向量。

除了上述简单的转置操作之外，Matlab 还提供了一个带前置点号的转置操作符。这时，可以把点-转置操作符解释成非复数共轭转置。当数组为复数数组时，转置（'）给出的结果是复共轭转置，也就是说，在进行转置操作过程中，虚部的符号也跟着改变了。相反，点-转置操作符（.'）只将数组转置，不进行共轭操作。

```
>> c = a.'
c =
     1
     2
     3
     4
     5
```

显然，对于实数而言，.' 和 ' 是等效的。

```
>> d = complex(a,a)
d =
   Columns 1 through 4
   1.0000 + 1.0000i   2.0000 + 2.0000i   3.0000 + 3.0000i   4.0000 + 4.0000i
   Column 5
   5.0000 + 5.0000i
```

这条命令利用函数 complex 由 a 生成一个简单的复数行向量。

```
>> e = d'
e =
    1.0000 - 1.0000i
    2.0000 - 2.0000i
    3.0000 - 3.0000i
    4.0000 - 4.0000i
    5.0000 - 5.0000i
```

这条命令生成一个列向量 e，它是 d 的复共轭转置向量。

```
>> f = d.'
f =
    1.0000 + 1.0000i
    2.0000 + 2.0000i
    3.0000 + 3.0000i
    4.0000 + 4.0000i
    5.0000 + 5.0000i
```

这条命令生成一个列向量 f，它是 d 的转置向量。

如果一个数组既可以是一个行向量，也可以是一个列向量，那么我们自然就会想到数组包含多个行和列。也就是说，数组还可以以矩阵的形式存在。矩阵的生成同样遵循行向量和列向量的生成规则：逗号或者空格用来隔开一行中的元素，分号则用来隔开列中的元素。如下例：

```
>> g = [1   2   3   4;5   6   7   8]
```

```
g =
    1    2    3    4
    5    6    7    8
```

这里，g是一个2行4列的数组或者矩阵；也就是说，它既是一个2×4的矩阵，又是一个维数为2和4的矩阵。语句中的分号告诉Matlab从5开始创建新行。在下面的语句中，请读者注意断行操作（回车）。

```
>> g = [1    2    3    4
        5    6    7    8
        9   10   11   12]
g =
    1    2    3    4
    5    6    7    8
    9   10   11   12
```

从上例我们可以看到，除了分号之外，在输入数组元素时按下回车键也会创建新行。

在创建多行向量或矩阵时，Matlab严格规定所有行的列数必须相同，下例的创建将会导致失败。

```
>> h = [1   2   3; 4    5   6   7]
??? Error using ==> vertcat
All rows in the bracketed expression must have the same
number of columns.
```

5.5 标量-数组运算

在本章最开始给出的例子中就涉及到了标量-数组运算，即数组x乘以标量π。标量和数组之间的其他简单数学运算也遵循与之相同的规则。数组与标量之间的加、减、乘、除都是简单地对数组中的所有元素进行运算。如下例：

```
>> g-2
ans =
    -1        0        1        2
     3        4        5        6
     7        8        9       10
```

这条命令从g的每一个元素中减去2。

```
>> 2*g - 1
ans =
     1        3        5        7
     9       11       13       15
    17       19       21       23
```

这条命令将g的每一个元素乘以2，然后再减去1。

```
>> 2*g/5 + 1
ans =
    1.4      1.8      2.2      2.6
      3      3.4      3.8      4.2
    4.6        5      5.4      5.8
```

这条命令将 g 的每一个元素乘以 2，然后除以 5，最后加 1。

值得注意的是，标量-数组运算使用了与标量表达式中相同的计算优先级来决定运算的先后顺序。

5.6　数组-数组运算

数组之间的数学运算就没有标量和数组之间的运算这么简单了。很明显，不同大小或不同维数数组之间的运算是难以定义的，并且具有不确定的结果。不过，当两个数组具有相同维数时，加法、减法、乘法和除法在 Matlab 中就可以逐元素（元素对元素）进行。例如：

```
>> g % recall previous array
g =
     1    2    3    4
     5    6    7    8
     9   10   11   12
>> h [1  1  1   1;2  2  2   2;3  3  3   3] % create new array
h =
     1    1    1    1
     2    2    2    2
     3    3    3    3
>> g + h % add h to g on an element-by-element basis
ans =
     2    3    4    5
     7    8    9   10
    12   13   14   15
>> ans - h % subtract h from the previous answer to get g back
ans =
     1    2    3    4
     5    6    7    8
     9   10   11   12
>> 2*g - h % multiplies g by 2 and subtracts h from the result
ans =
     1    3    5    7
     8   10   12   14
    15   17   19   21
>> 2*(g - h) % use parentheses to change order of operation
ans =
     0    2    4    6
     6    8   10   12
    12   14   16   18
```

注意：数组-数组运算也使用了与标量表达式中相同的计算优先级来决定运算的先后顺序，并且可以利用括号来随意改变运算顺序。

元素对元素的乘法和除法与普通的乘除法类似，但是使用了并不怎么常用的符号：

```
>> g.*h
ans =
```

```
     1          2          3          4
    10         12         14         16
    27         30         33         36
```

这里，我们用点乘符号“.*”对 g 和 h 进行了元素对元素的乘法。

> 标准的星号乘法符号之前的这个点号告诉 Matlab 执行元素对元素的数组乘法。没有这个点号的乘法表明进行的是矩阵乘法，关于矩阵乘法的讨论将会在后边给出。

针对上面例子的矩阵乘法是未定义的：

```
>> g*h
??? Error using ==> *
Inner matrix dimensions must agree.
```

元素对元素的除法，或者说点除，也需要用到点号，例如：

```
>> g./h
ans =
    1.0000  2.0000  3.0000  4.0000
    2.5000  3.0000  3.5000  4.0000
    3.0000  3.3333  3.6667  4.0000
>> h.\g
ans =
    1.0000  2.0000  3.0000  4.0000
    2.5000  3.0000  3.5000  4.0000
    3.0000  3.3333  3.6667  4.0000
```

与标量除法相同，Matlab 用斜线（/）和反斜线（\）定义数组除法。在这两种定义中，都是用斜线右边的矩阵去除斜线左边的矩阵，即左边是分子，右边是分母。

> 在斜线和反斜线前边的点号告诉 Matlab 执行元素对元素的数组除法。没有点号的除法表明进行的是矩阵除法。关于矩阵除法的讨论，将会在后边给出。

在分子是标量的情况下，数组除法或者点除同样适用，例如：

```
>> 1./g
ans =
          1              0.5          0.33333             0.25
        0.2          0.16667          0.14286            0.125
    0.11111              0.1         0.090909         0.083333
```

在这个例子中，分子中的标量 1 被扩展成一个和分母相同维数的数组，然后执行元素对元素的矩阵除法。也就是说，上例给出的是下面两行命令的简易表达方式：

```
>> f=[1  1  1   1;1  1  1   1;1  1  1   1] % create numerator
f =
    1    1    1    1
    1    1    1    1
    1    1    1    1
>> f./g
```

```
ans =
         1           0.5       0.33333        0.25
       0.2       0.16667       0.14286       0.125
   0.11111           0.1      0.090909    0.083333
>> f./h
ans =
         1             1             1           1
       0.5           0.5           0.5         0.5
   0.33333       0.33333       0.33333     0.33333
```

上面自动扩展标量值使得元素对元素的运算能够进行的过程称为标量扩展。标量扩展在 Matlab 中得到了广泛的应用。

没有点号的除法是矩阵除法运算，或称为矩阵逆运算，这是一个与标量运算完全不同的运算，如下例：

```
>> g/h
Warning: Rank deficient, rank = 1 tol = 5.3291e-015.
ans =
        0        0      0.83333
        0        0       2.1667
        0        0          3.5
>>h/g
Warning: Rank deficient, rank = 2  tol = 1.8757e-014.
ans =
     -0.125      0      0.125
      -0.25      0       0.25
     -0.375      0      0.375
```

矩阵除法给出的结果并不一定和矩阵 g、h 具有相同的维数。(有关矩阵的运算请参见第 17 章。)

在 Matlab 中，数组的指数运算有好几种定义方式。与乘法和除法一样，^是专门为矩阵指数运算保留的，而.^是用来执行元素对元素的指数运算的。当指数是一个标量时，该标量将用来对数组的所有元素进行取指数操作。如下例：

```
>> g,h % recalls the arrays used earlier
g =
    1    2    3    4
    5    6    7    8
    9   10   11   12
h =
    1    1    1    1
    2    2    2    2
    3    3    3    3
>> g.^2
ans =
     1      4      9     16
    25     36     49     64
    81    100    121    144
```

该语句取数组 g 的所有元素的平方值，但下面的语句就会出错：

```
>> g^2
??? Error using ==> mpower
Matrix must be square.
```

因为上面的语句是矩阵指数运算，而此运算只对方阵（行数和列数相同的矩阵）才有定义。

```
>> g.^-1
ans =
          1            0.5        0.33333         0.25
        0.2        0.16667        0.14286        0.125
    0.11111            0.1       0.090909     0.083333
```

该语句得到数组 g 的每个元素的倒数。

```
>> 1./g
ans =
          1            0.5        0.33333         0.25
        0.2        0.16667        0.14286        0.125
    0.11111            0.1       0.090909     0.083333
```

该语句得到的结果和先前的标量扩展法得到的结果是一样的。

当一个标量的指数是矩阵时，矩阵的每个元素都被应用到这个标量上进行取指数运算。例如：

```
>> 2.^g
ans =
     2       4       8      16
    32      64     128     256
   512    1024    2048    4096
```

这条语句将得到一个与 g 同维的矩阵，矩阵的每一个元素即为 2 的 g_{ij} 次幂，其中 g_{ij} 是矩阵 g 中的元素值。

如果指数运算中的两个变量（指数和底数）是相同维数的数组，就执行元素对元素的指数运算。例如：

```
>> g.^h
ans =
     1       2       3       4
    25      36      49      64
   729    1000    1331    1728
```

这条命令将得到与 g 和 h 相同维数的矩阵，矩阵中的每一个元素即为 g_{ij} 的 h_{ij} 次幂，其中，g_{ij} 和 h_{ij} 分别是矩阵 g 和 h 中的元素值。上例中，结果矩阵的第一行没有发生变化，这是因为 h 的第一行都是 1（取 1 次幂）；第二行取平方值；第三行取立方值。

```
>> g.^(h-1)
ans =
     1       1       1       1
     5       6       7       8
    81     100     121     144
```

这条语句表明标量和数组运算可以联合进行。

带有标量参数的两种指数运算形式同样也是标量扩展的特例。我们很自然就能想到这样的运算实际上就是，先把标量扩展成和数组相同的维数，然后再执行元素对元素的指数运算。

下面这个表格总结了基本的数组运算。

元素对元素运算	数据样例 $A = [a_1\ a_2 \dots a_n]$，$B = [b_1\ b_2 \dots b_n]$，c = <一个标量>
标量加法	$A+c = [a_1+c\ \ a_2+c\ \dots a_n+c]$
标量减法	$A-c = [a_1-c\ \ a_2-c\ \dots a_n-c]$
标量乘法	$A*c = [a_1*c\ \ a_2*c\ \dots a_n*c]$
标量除法	$A/c = c\backslash A = [a_1/c\ \ a_2/c\ \dots a_n/c]$
数组加法	$A+B = [a_1+b_1\ \ a_2+b_2 \dots\ a_n+b_n]$
数组乘法	$A.*B = [a_1*b_1\ \ a_2*b_2 \dots\ a_n*b_n]$
数组右除	$A./B = [a_1/b_1\ \ a_2/b_2 \dots\ a_n/b_n]$
数组左除	$A.\backslash B = [a_1/b_1\ \ a_2/b_2 \dots\ a_n/b_n]$
数组乘方	$A.\hat{}c = [a_1\hat{}c\ \ a_2\hat{}c \dots\ a_n\hat{}c]$ $c.\hat{}A = [c\hat{}\ a_1\ \ c\hat{}a_2 \dots\ c\hat{}a_n]$ $A.\hat{}B = [a_1\hat{}\ b_1\ \ a_2\hat{}\ b_2 \dots\ a_n\hat{}\ b_n]$

5.7 标准数组

鉴于标准数组的通用性，Matlab 专门提供了一些函数来创建它们。标准数组通常包括全 1 数组、全 0 数组、单位矩阵、随机矩阵、对角矩阵以及元素为指定常数的数组。下面的命令将创建全 0 和全 1 数组：

```
>> ones(3)
ans =
     1          1          1
     1          1          1
     1          1          1
>> zeros(2,5)
ans =
     0          0          0          0          0
     0          0          0          0          0
>>size(g)
ans =
     3          4
>> ones(size(g))
ans =
     1          1          1          1
     1          1          1          1
     1          1          1          1
```

对于 ones 和 zeros 函数，当只有一个输入参数时，即 ones(n)或 zeros(n)，Matlab 就分别生成一个 n×n 的全 1 或者全 0 数组；当有两个输入参数时，即 ones(r,c)或者 zeros(r,c)，Matlab 就分别生成一个 r 行 c 列的全 1 或者全 0 数组。要想生成一个与其他数组相同维数的全 1 或者全 0 数组，用户只要在 ones 或者 zeros 的参数中调用 size 函数（这个函数将在本章的后边讨论）就可以了。

下面的命令将创建单位矩阵：

```
>> eye(4)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
>> eye(2,4)
ans =
     1     0     0     0
     0     1     0     0
>> eye(4,2)
ans =
     1     0
     0     1
     0     0
     0     0
```

如上所示，函数 eye 用与生成全 1 数组和全 0 数组相同的语法格式来生成单位矩阵。单位矩阵或数组是具有如下取值的矩阵或数组：除 A(i,i)之外，所有其他元素都为 0，其中 i=1:min(r,c)，min(r,c)是矩阵 A 的行数和列数的最小值。

下面的命令将创建随机矩阵：

```
>> rand(3)
ans =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214
>> rand(1,5)
ans =
    0.4447    0.6154    0.7919    0.9218    0.7382
>> b = eye(3)
b =
     1     0     0
     0     1     0
     0     0     1
>> rand(size(b))
ans =
    0.1763    0.9169    0.0579
    0.4057    0.4103    0.3529
    0.9355    0.8937    0.8132
```

函数 rand 生成均匀分布的随机数组，其元素取值介于 0~1 之间。另外，还有一个函

数 randn 将生成均值为 0、方差为 1 的正态分布矩阵，如下例：

```
>> randn(2)
ans =
   -0.4326      0.1253
   -1.6656      0.2877
>> randn(2,5)
ans =
   -1.1465      1.1892      0.3273      -0.1867      -0.5883
    1.1909     -0.0376      0.1746       0.7258        2.183
```

函数 diag 生成对角数组，在该数组中，一个向量可以被放在与数组的主对角线平行的任何位置上，如下例：

```
>> a = 1:4 % start with a simple vector
a =
    1     2     3     4
>> diag(a) % place elements on the main diagonal
ans =
    1     0     0     0
    0     2     0     0
    0     0     3     0
    0     0     0     4
>> diag(a,1) % place elements 1 place up from diagonal
ans =
    0     1     0     0     0
    0     0     2     0     0
    0     0     0     3     0
    0     0     0     0     4
    0     0     0     0     0
>> diag(a,-2) % place elements 2 places down from diagonal
ans =
    0     0     0     0     0     0
    0     0     0     0     0     0
    1     0     0     0     0     0
    0     2     0     0     0     0
    0     0     3     0     0     0
    0     0     0     4     0     0
```

利用上述标准数组，我们可以利用几种不同的方式来生成一个所有元素值都相同的数组。如下例：

```
>> d = pi; % choose pi for this example
>> d*ones(3,4) % slowest method (scalar-array multiplication)
ans =
        3.1416      3.1416      3.1416      3.1416
        3.1416      3.1416      3.1416      3.1416
        3.1416      3.1416      3.1416      3.1416
>> d+zeros(3,4) % slower method (scalar-array addition)
ans =
        3.1416      3.1416      3.1416      3.1416
```

```
        3.1416      3.1416      3.1416      3.1416
        3.1416      3.1416      3.1416      3.1416
>> d(ones(3,4)) % fast method (array addressing)
ans =
        3.1416      3.1416      3.1416      3.1416
        3.1416      3.1416      3.1416      3.1416
        3.1416      3.1416      3.1416      3.1416
>> repmat(d,3,4) % fastest method (optimum array addressing)
ans =
        3.1416      3.1416      3.1416      3.1416
        3.1416      3.1416      3.1416      3.1416
        3.1416      3.1416      3.1416      3.1416
```

对于小数组而言，上边这些方法均可取。但是，随着数组维数的增大，含有标量乘法的方法（d*ones(r,c)）就会使矩阵生成过程变慢。因为加法通常都比乘法运算速度快，所以较好的办法就是将用到的标量加到一个全 0 数组上（d+zeros(r,c)）。尽管后两种方法不是那么直观，但它们却是生成大数组的最快方法。因为它们都用到了前边所讲过的数组索引。

第三种方法（d(ones(r,c))）先生成一个 r×c 的全 1 数组，然后用这个数组来索引和复制标量 d。尽管这种方法没有用到浮点运算，但生成一个全 1 临时数组将会占用内存，并且消耗时间，因此使得这种方法的速度变慢。方法 repmat(d,r,c)调用函数 repmat，即 replicate matrix（复制矩阵）的缩写。对于标量 d，该函数执行如下操作步骤：

```
D(r*c) = d;           % a row vector whose (r*c)-th element is d
D(:)= d;              % scalar expansion to fill all elements of D with d
D = reshape(D,r,c);   % reshape the vector into the desired r-by-c shape
```

上边的 Matlab 代码首先采用标量扩展方法生成一个具有 r×c 个元素的向量，该向量的所有元素均为 d；然后用函数 reshape 将这个向量变换成一个 r×c 的数组。（后边我们将会对函数 repmat 和 reshape 做进一步讨论。）

5.8 数组处理方法

数组是 Matlab 的基础，其处理方法也多种多样。数组生成以后，Matlab 就通过指定特定的脚标，提供插入、提取和重排数组子集的功能强大的方法。了解这些处理特性有助于用户更有效地使用 Matlab。让我们先通过几个示例看一下 Matlab 的数组处理特性：

```
>> A = [1   2   3;4   5   6;7   8   9]
A =
    1       2       3
    4       5       6
    7       8       9
>> A(3,3) = 0 % set element in 3rd row, 3rd column to zero
A =
    1       2       3
    4       5       6
    7       8       0
```

该命令将数组 A 的第 3 行第 3 列的元素值变为 0。

```
>> A(2,6) = 1 % set element in 2nd row, 6th column to one
A =
     1     2     3     0     0     0
     4     5     6     0     0     1
     7     8     0     0     0     0
```

该语句在数组的第 2 行第 6 列置入 1。因为 A 没有 6 列，数组 A 的维数就根据需要增加了，并且在其他没有赋值的位置填上了 0，以便使数组保持为一个矩形矩阵。

```
>> A(:,4) = 4
A =
     1     2     3     4     0     0
     4     5     6     4     0     1
     7     8     0     4     0     0
```

该语句将数组 A 的第 4 列设置为 4。因为 4 是一个标量，Matlab 就将它扩展以便将所有指定位置的元素都填充为 4。这是标量扩展的另一个示例。Matlab 进行标量扩展是为了简化那些虽然明确但却繁琐的语句。例如，上面的语句等效于下面这条稍显复杂的语句：

```
>> A(:,4) = [4;4;4]
A =
     1     2     3     4     0     0
     4     5     6     4     0     1
     7     8     0     4     0     0
>> A(:,4) = [4 4 4] % but a row can't be squeezed into a column!
??? In an assignment A(:,matrix) = B, the number of elements in the
subscript of A and the number of columns in B must be the same.
```

让我们重新调用 A 的初始值，再看看其他一些数组处理方法。

```
>> A = [1 2 3;4 5 6;7 8 9]; % restore original data
>> B = A(3:-1:1,1:3)
B =
     7     8     9
     4     5     6
     1     2     3
>> B = A(end:-1:1,1:3) % same as above
B =
     7     8     9
     4     5     6
     1     2     3
```

该语句通过将数组 A 的行按逆序排列，得到数组 B。关键字 end 自动指向指定维数的最后或最大索引。本例中，end 指向最大的行索引 3。

```
>> B = A(3:-1:1,:)
B =
     7     8     9
     4     5     6
     1     2     3
```

该语句实现的功能与前一条语句一样。其中，最后一个冒号表示对所有的列进行操作。也就是说，在该例中，此冒号与 1:3 等价，因为 A 只有 3 列。

```
>> C = [A B(:,[1 3])]
C =
     1     2     3     7     9
     4     5     6     4     6
     7     8     9     1     3
```

该语句通过将数组 B 的第一和第三列附加或连接在数组 A 的右侧，从而生成数组 C。

```
>> B = A(1:2,2:3)
B =
     2     3
     5     6
>> B = A(1:2,2:end) % same as above
B =
     2     3
     5     6
```

这两条语句通过提取矩阵 A 的前两行后两列的元素来生成矩阵 B。在这里，再一次用到了冒号表示法，用于生成待提取的数组元素的索引向量。在第二个例子中，end 表示最后或最大的列索引号。

```
>> C = [1 3]
C =
     1     3
>> B = A(C,C)
B =
     1     3
     7     9
```

上例用数组 C 作为数组 A 的索引，而不是直接用冒号表示法（start:increment:end 或 start:end）直接指定数组 A 中的元素。在这个例子中，B 是由数组 A 的第 1 行第 1 列、第 1 行第 3 列、第 3 行第 1 列和第 3 行第 3 列的元素构成的。

```
>> B = A(:)
B =
     1
     4
     7
     2
     5
     8
     3
     6
     9
```

该语句通过依次提取数组 A 的各列，将数组 A 延展成一个列向量 B。这种方法是把一个数组重构成另一个不同维数、但数组元素完全相同的数组的最简单方式。

```
>> B = B. '
B =
     1     4     7     2     5     8     3     6     9
>> B = reshape(A,1,9) % reshape A into 1-by-9
B =
     1     4     7     2     5     8     3     6     9
>> B = reshape(A,[1 9])
B =
     1     4     7     2     5     8     3     6     9
```

这些命令说明了先前介绍过的点-转置操作符以及函数 reshape 的使用。在这个例子中，reshape 函数中的索引参数可以是单独的函数参数（reshape(A,1,9)），或者是一个向量参数（reshape(A,[1 9])）。

```
>> B = A % copy A into B
B =
     1     2     3
     4     5     6
     7     8     9
>> B(:,2) = []
B =
     1     3
     4     6
     7     9
```

该命令通过去除原来数组 B 中的第二列重新定义数组 B。如果数组的某个部分被设置成空矩阵或者空数组[ ]，这部分将被删除，原数组将缩维成由剩余元素构成的新数组。需要注意的是，数据必须被整行或整列地删除，这样才能保证缩维后的数组仍旧是一个矩形。

```
>> C = B.'
C =
     1     4     7
     3     6     9
>> reshape(B,2,3) % reshape is not equivalent to transpose
ans =
     1     7     6
     4     3     9
```

上述命令显示了一个数组的转置，并且说明了 reshape 函数和转置的不同之处。转置将原矩阵的第 i 行元素转置成结果矩阵的第 i 列元素，因此原来的 3×2 的数组被转置成一个 2×3 的数组。

```
>> C(2,:) = []
C =
     1     4     7
```

该语句去除了 C 的第 2 行元素，剩下了一个行向量。

```
>> A(2,:) = C
A =
     1     2     3
```

```
    1        4        7
    7        8        9
```

该命令用数组 C 取代了数组 A 的第 2 行。

```
>> B = A(:,[2 2 2 2]) % create new B array
B =
    2        2        2        2
    4        4        4        4
    8        8        8        8
>> B = A(:,2+zeros(1,4)) % [2 2 2 2]=2+zeros(1,4)
B =
    2        2        2        2
    4        4        4        4
    8        8        8        8
>> B = repmat(A(:,2),1,4) % replicate 2nd column into 4 columns
B =
    2        2        2        2
    4        4        4        4
    8        8        8        8
```

上述命令用 3 种方式将数组 A 的第 2 列复制 4 次而得到数组 B。对于大数组而言，最后一种方法的速度最快。

```
>> A, C % show A and C again
A =
    1        2        3
    1        4        7
    7        8        9
C =
    1        4        7
>> A(2,2) = []
??? Indexed empty matrix assignment is not allowed.
```

上述命令表明，用户只能整行或者整列地删除数组中的元素。在只有行或列中的部分元素被删除时，Matlab 无法对数组进行缩维操作。

```
>> C = A(4,:)
??? Index exceeds matrix dimensions.
```

由于 A 没有第 4 列，Matlab 无法按照命令进行操作，并且给出了出错信息。以这个结果为例，下面给出了数组索引必须遵循的原则。

> 如果 A(r,c) 出现在等号的左边，并且用 (r,c) 声明的一个或者多个元素并不存在，就根据需要将 A 扩展，并将扩展位置上的元素置为 0，这样就使得 A(r,c) 均指向已知元素。但是，如果数组 A 出现在等号右边，那么 A(r,c) 所指向的所有元素都必须存在，否则就返回一个出错信息。

继续来看下面的例子：

```
>> C(1:2,:) = A
```

```
??? Subscripted assignment dimension mismatch.
```

该命令表明用户不能将一个数组置入另一个不同维数的数组中。

```
>> C(3:4,:) = A(2:3,:)
C =
     1     4     7
     0     0     0
     1     4     7
     7     8     9
```

上例显示用户可以把数组 A 的第 2 和第 3 列元素置入到数组 C 的相同大小的区域中。因为数组 C 的第 2 到第 4 行并不存在，Matlab 就根据需要自动地生成了这些行。另外，数组 C 的第 2 行没有被指定，因此这一行被全部填 0。

```
>> A = [1 2 3;4 5 6;7 8 9] % fresh data
A =
     1     2     3
     4     5     6
     7     8     9
>> A(:,2:3) % a peek at what's addressed next
ans =
     2     3
     5     6
     8     9
>> G(1:6) = A(:,2:3)
G =
     2     5     8     3     6     9
```

该命令通过提取数组 A 的第 2 列和第 3 列元素，并依次排列而生成行向量 G。需要注意的是，等号两侧的这些矩阵的形状是不同的。通过将数组 A 的第 1 列和第 2 列元素分别从上到下依次赋值给数组 G，数组 A 的元素就被插入到了数组 G 中。

```
>> H = ones(6,1);        % create a column array

>> H(:) = A(:,2:3)       % fill H without changing its shape
H =
     2
     5
     8
     3
     6
     9
```

当（:）出现在等号左边时，意味着从等号右侧提取相应的元素，然后原封不动地放到左边的数组中。在上述例子中，从数组 A 中提取第 2 列和第 3 列，然后将它们插入到列向量 H 中。很明显，上边的命令要正常运行，等号两边所引用的元素数目必须相同。

当等号的右边是一个标量，左边是一个数组时，就需要用到标量扩展来填充。例如：

```
>> A(2,:) = 0
A =
```

```
     1         2         3
     0         0         0
     7         8         9
```

该命令将数组 A 的第 2 行全部替换成 0。我们发现，等号右边的单个 0 经扩展后填充到左边索引所指定的位置。该例的执行效果与下面的代码相同：

```
>> A(2,:) = [0 0 0]
A =
     1         2         3
     0         0         0
     7         8         9
```

在任何情况下，只要在该调用数组的地方使用了标量，就会出现标量扩展。这时，Matlab 自动将标量进行扩展并填充到指定位置，然后再执行用户下达的指令。

下面是标量扩展的另一种情况：

```
>> A(1,[1 3]) = pi
A =
    3.1416    2.0000    3.1416
         0         0         0
    7.0000    8.0000    9.0000
```

在该例中，标量 π 被扩展后填进了两个位置。现在我们回顾一下函数 reshape 在标量输入的情况下是如何工作的。首先创建一个含有数字 2 的 2×4 的数组：

```
>> D(2*4) = 2 % create array with 8 elements
D =
     0         0         0         0         0         0         0         2
>> D(:) = 2 % scalar expansion
D =
     2         2         2         2         2         2         2         2
>> D = reshape(D,2,4) % reshape
D =
     2         2         2         2
     2         2         2         2
```

第一行命令（D(2*4)=2）生成一个长度为 8 的行向量，并将最后一列置为 2。第二行命令（D(:)=2）利用标量扩展将 D 的所有元素填充为 2。第三行命令（D=reshape(D,2,4)）将上一步的结果转换成指定维数的数组。

有时候，我们需要执行向量和二维数组（矩阵）之间的数学运算。例如，我们给定一个矩阵 A 和一个向量 r：

```
>> A = reshape (1:12,3,4)'
A =
      1         2         3
      4         5         6
      7         8         9
     10        11        12
>> r = [3 2 1]
r =
       3        2         1
```

假设我们希望将数组 A 的第 i 列元素减去 r(i)。其中的一种实现方法是：

```
>> Ar = [A(:,1)-r(1) A(:,2)-r(2) A(:,3)-r(3)]
Ar =
    -2       0        2
     1       3        5
     4       6        8
     7       9       11
```

另外，还可以利用索引来实现：

```
>> R = r([1 1 1 1],:) % duplicate r to have 4 rows
R =
    3       2       1
    3       2       1
    3       2       1
    3       2       1
>> Ar = A - R % now use element by element subtraction
Ar =
    -2      0       2
     1      3       5
     4      6       8
     7      9      11
```

其实，获得矩阵 R 更快更常用的方法是利用函数 ones 和 size，或者利用函数 repmat，具体实现方法如下：

```
>> R = r(ones(size(A,1),1),:) % historically this is Tony's trick
R =
    3       2       1
    3       2       1
    3       2       1
    3       2       1
>> R = repmat(r,size(A,1),1) % often faster than Tony's trick
R =
    3       2       1
    3       2       1
    3       2       1
    3       2       1
```

该例中，size(A,1)返回数组 A 的行数。

有时候，只用一个索引来寻找数组元素更加方便。在 Matlab 中，当只使用一个索引时，该索引值的计数方式为：从第一列开始，从上到下逐行计数，然后按此方式逐列进行。例如：

```
>> D = reshape(1:12,3,4) % new data
D =
    1       4       7       10
    2       5       8       11
    3       6       9       12
>> D(2)
```

```
ans =
    2
>> D(5)
ans =
    5

>> D(end)
ans =
    12
>> D(4:7)
ans =
    4        5        6        7
```

Matlab 函数 sub2ind 将一个单一索引值转换成行列下标，ind2sub 则将行列下标转换成单一索引值。如下例：

```
>> sub2ind(size(D),2,4) % find single index from row and column
ans =
    11
>> [r,c] = ind2sub(size(D),11) % find row and column from single index
r =
    2
c =
    4
```

上例表明 D 的第 2 行第 4 列的元素即是第 11 个元素。需要注意的是，这两个函数都需要知道它所进行转换的数组的维数，也就是 size（D），无需知道数组 D 本身。

> 下标指数组元素的行列位置，例如，A(2,3)指 A 的第二行第三列元素。数组元素的索引指相对于第一行第一列那个元素（其索引为 1）的位置。索引顺序是先逐行计数，再按列递进。也就是说，索引是按照下标出现的顺序（即先行后列）计数的，例如，如果 D 有 3 行，则 D（8）为其第三列的第二个元素。

除了用基于下标的方式寻址数组外，逻辑运算（本书后面将会进行详细讨论）所得到的逻辑数组也可以用来寻址数组，条件是逻辑数组的维数要和寻址数组的维数一致。在利用逻辑数组进行寻址时，与 True（1）相对应的元素将被保留下来，而与 False（0）相对应的元素将被删除。为了便于理解，我们来看下面的例子：

```
>> x = -3:3 % Create data
x =
    -3      -2      -1       0       1       2       3
>> abs(x)>1
ans =
     1       1       0       0       0       1       1
```

该命令返回一个逻辑数组，在 x 的元素的绝对值大于 1 的地方，返回 1（True），在绝对值小于等于 1 的地方，返回 0（False）。有关逻辑表达式的更详细内容请参见本书关于逻辑表达式的章节部分。

```
>> y = x(abs(x)>1)
y =
    -3       -2       2       3
```

该命令生成了一个数组 y，其元素值就是在 x 中绝对值大于 1 的那些元素的值。

但是，需要注意的是，尽管 abs(x)>1 生成了数组[1 1 0 0 0 1 1]，但它并不等价于包含这些值的数字数组，下面的语句将会出错：

```
>> y = x([1 1 0 0 0 1 1])
??? Subscript indices must either be real positive integers or logicals.
```

该命令出错原因在于，尽管 abs(x)>1 和[1 1 0 0 0 1 1]看上去是相同的向量，但一个是逻辑数组，一个是数字数组，二者迥然不同。我们可以利用 class 函数观察二者之间的差异：

```
>> class(abs(x)>1)   % logical result from logical comparison
ans =
logical
>> class([1 1 0 0 0 1 1])   % double precision array
ans =
double
```

另外，利用函数 islogical 和 isnumeric 可以检测出数组是否是逻辑数组或数字数组，两个函数均返回逻辑值 True（1）或 False（0），例如：

```
>> islogical(abs(x)>1)
ans =
    1
>> islogical([1 1 0 0 0 1 1])
ans=
    0
>> isnumeric(abs(x)>1)
ans=
    0
>> isnumeric([1 1 0 0 0 1 1])
ans=
    1
```

因此 Matlab 试图去指向在[1 1 0 0 0 1 1]中声明的元素编号所对应的元素，这样就出现了一个错误，因为在位置 0 并没有一个元素存在。

Matlab 提供了一个函数 logical 来将数字数组转换成逻辑数组：

```
>> y = x(logical([1 1 0 0 0 1 1]))
y =
    -3       -2       2       3
```

我们又一次得到了所需要的结果。逻辑数组是 Matlab 的另一种数据类型。到目前为止，我们讨论的都是数字数组。现在总结如下：

> 用数字数组指定数组下标就提取出具有给定数字索引的元素。另一方面，用逻辑表达式和函数 logical 返回的逻辑数组指定数组下标，就提取出值为逻辑真（1）的元素。

在实数域中，我们通常使用ones和zeros函数来创建数组，在逻辑域中，我们则可以使用true和false函数来分别创建由True和False值组成的逻辑数组，如下面的代码所示：

```
>> true
ans =
    1
>> true(2,3)
ans =
    1       1       1
    1       1       1
>> false
ans =
    0
>> false(1,6)
ans =
    0       0       0       0       0       0
```

上述结果从表面上看好像与 ones 和 zeros 函数返回的值相同，实际上它们都是逻辑值，而非数值。

逻辑数组对二维数组及向量同样有用，例如：

```
>> B = [5 -3;2 -4] % new data
B =
    5       -3
    2       -4
>> x = abs(B)>2 % logical result
x =
    1       1
    0       1
>> y = B(x) % grab True values
y =
     5
    -3
     4
```

但是，上边的这个最后结果是作为一个列向量返回的，这是因为没有办法定义一个只有 3 个元素的二维数组。不论用户提取了多少个元素，Matlab 都用单一索引顺序来提取所有为真的元素，然后将提取的结果形成一个列向量。

上边所用到的数组寻址技术可以用下表来总结：

数组寻址	描述
A(r,c)	用所希望的行 r 和所希望的列 c 定义的索引向量来寻址 A 的子数组
A(r,:)	用所希望的行 r 和所有的列所定义的索引向量来寻址 A 的子数组
A(:,c)	用所有的行和所希望的列 c 所定义的索引向量来寻址 A 的子数组
A(:)	以列向量的方式依次寻址 A 的所有元素，依列进行。如果 A(:)出现在等号的左侧，这意味着用等号右侧的元素来填充数组 A，而不用改变 A 的形状
A(k)	用单一索引向量 k 寻址 A 的子数组，就像 A 是列向量 A(:)一样

（续表）

数组寻址	描述
A(x)	用逻辑数组 x 寻址 A 的子数组。注意：x 必须与 A 有相同的维数。如果 x 的维数小于 A，则 x 中缺少的那一部分将被假设为 False；如果 x 的维数大于 A，则 x 中多余的那一部分必须被设置为 False

5.9　数组排序

在很多时候，我们都需要对一个给定的数据向量进行排序。为了完成这一操作，Matlab 提供了 sort 函数，该函数将任意给定的序列按升序排列。例如，下面的代码首先生成一个随机排列的向量 x，然后利用 sort 函数对其进行排序：

```
>> x = randperm(8) % new data
x =
     7     5     2     1     3     6     4     8
```

（注意：上述程序中。由于 randperm 函数具有随机性，因此，用户在验证时所得的结果可能与本书不一致。）

```
>> xs = sort(x)  % sort ascending by default
xs =
     1     2     3     4     5     6     7     8
>> xs = sort(x,'ascend') % sort ascending
xs =
     1     2     3     4     5     6     7     8
>> [xs,idx] = sort(x) % return sort index as well
xs =
     1     2     3     4     5     6     7     8
idx =
     4     3     5     7     2     6     1     8
```

由上面的代码结果可知，当只有一个输出变量时，函数 sort 将返回向量排序后的结果；当有两个输出变量时，sort 函数在第一个变量输出向量排序后的结果，在第二个变量输出排序结果的索引。这两个变量之间的关系是 xs(k)=x(idx(k))。

> 注意，当一个 Matlab 函数具有两个或者更多的输出变量时，这些变量必须用逗号隔开，并放在等号左边的方括号内。我们在前面介绍数组处理时，也看到过类似的写法，但在那里我们利用方括号将两个数组连接起来，而不是用于函数的返回值。另外，用于函数输出的方括号语法出现在等号左边，而用于数组连接的方括号语法出现在等号右边。

在过去的 Matlab 版本中，要想使一个数组按降序排列，必须使用数组索引技术将 sort 函数的输出结果进行调换。例如，下面的代码利用数组索引技术实现了 x 的降序排列：

```
>> xsd = xs(end:-1:1)
xsd =
```

```
    8     7     6     5     4     3     2     1
>> idxd = idx(end:-1:1)
idxd =
    8     1     6     2     7     5     3     4
```

在 Matlab 7 中，sort 函数本身已能够执行降序排列操作。这时只要再给 sort 函数添加一个输入参数'descend'即可。例如，下面的代码利用 sort 函数实现了 x 的降序排列：

```
>> xs = sort(x,'descend')  % sort descending
xs =
    8     7     6     5     4     3     2     1
```

如果要排序的数组是一个二维数组，则 sort 函数只对该数组的列进行排序。下面的代码实现了对二维数组 A 的排序：

```
>> A = [randperm(6);randperm(6);randperm(6);randperm(6)] % new data
A =
    1     2     5     6     4     3
    4     2     6     5     3     1
    2     3     6     1     4     5
    3     5     1     2     4     6
>> [As,idx] = sort(A)
As =
    1     2     1     1     3     1
    2     2     5     2     4     3
    3     3     6     5     4     5
    4     5     6     6     4     6
idx =
    1     1     4     3     2     2
    3     2     1     4     1     1
    4     3     2     2     3     3
    2     4     3     1     4     4
```

其中，As 为 sort 函数对 A 的每一列进行升序排序后的结果，idx 为对应的排序后的索引。在很多情况下，用户可能只关心数组中某一列的排序问题。例如，下面的代码仅对 A 的第四列进行排序：

```
>> [tmp,idx] = sort(A(:,4)); % sort 4-th column only

>> As = A(idx,:) % rearrange rows in all columns using idx
As =
    2     3     6     1     4     5
    3     5     1     2     4     6
    4     2     6     5     3     1
    1     2     5     6     4     3
```

从结果可以看出，As 除了第四列按照升序排列外，其余各列与原数组 A 相同。

用户也可以利用 sort 函数对数组的各行进行排序，这时需要在待排序数组后面为 sort 函数提供一个附加的输入参数 2。例如，下面的代码分别对数组 A 按行的方向和按列的方向进行排序：

```
>> As = sort(A,2) % sort across 2-nd dimension
As =
     1     2     3     4     5     6
     1     2     3     4     5     6
     1     2     3     4     5     6
     1     2     3     4     5     6
>> As = sort(A,1) % same as sort(A)
As =
     1     2     1     1     3     1
     2     2     5     2     4     3
     3     3     6     5     4     5
     4     5     6     6     4     6
```

在默认情况下，如果用户不给 sort 函数提供第二个参数，则 sort 函数将按列的方向进行排序。当给 sort 函数传递第 2 个参数时，若该参数为 1，则按列的方向进行排序，若该参数为 2，则按行的方向进行排序。上述规定也是很容易理解的，因为在数组 A(r,c)中，先出现的是行，因此 sort(A,1)就意味着首先对一列的每一行元素进行排序；同样，因为在 A(r,c)中后出现的是列，因此 sort(A,2)就意味着对一行的每一列元素进行排序。虽然上面的代码中没有演示，'descend'也可以出现在 sort 函数调用中，例如 sort(A,1,'descend')，用于返回按降序排列的数组。（本书第 18 章将对 sort 函数进行更详细的阐述。）

5.10 子数组搜索

很多时候，用户都希望知道满足某些关系表达式的数组元素位于数组的什么位置。Matlab 提供了 find 函数实现这一功能，find 函数接受一个关系表示作为输入参数，返回满足这个表达式的所有数组元素的位置索引值。例如，下面的代码在向量 x 中寻找绝对值大于 1 的元素，并将这些元素组成向量 y：

```
>> x = -3:3
x =
    -3    -2    -1     0     1     2     3
>> k = find(abs(x)>1) % finds those subscripts where abs(x)>1
k =
     1     2     6     7
>> y = x(k) % creates y using the indices in k.
y =
    -3    -2     2     3
>> y = x(abs(x)>1) % creates the same y vector by logical addressing
y =
    -3    -2     2     3
```

find 函数也可以对二维数组进行操作。例如，下面的代码寻找数组 A 中所有大于 5 的元素的位置：

```
>> A = [1 2 3;4 5 6;7 8 9] % new data
A =
     1     2     3
     4     5     6
```

```
     7       8       9
>> [i,j] = find(A>5) % i and j are not equal to sqrt(-1) here
i =
     3
     3
     2
     3
j =
     1
     2
     3
     3
```

上例中的 i 和 j 中分别保存满足关系表达式 A>5 的元素的行和列的索引值。从结果可以看出，共有 4 个元素满足条件，即 A(3,1)、A(3,2)、A(2,3)和 A(3,3)。

对于二维数组，如果只给 find 函数提供一个输出变量，则 find 函数将返回满足条件的所有元素的单一索引值。例如，上面的例子可以改为：

```
>> k = find(A>5)
k =
     3
     6
     8
     9
```

有时候，使用二维数组的单一索引（如 A(k)）要比双值索引（如 A(i,j)）更加方便，也更容易理解。请看下面的例子：

```
>> A(k) % look at elements greater than 5
ans =
     7
     8
     6
     9

>> A(k) = 0 % set elements addressed by k to zero
A =
     1       2       3
     4       5       0
     0       0       0
>> A = [1 2 3;4 5 6;7 8 9] % restore data
A =
     1       2       3
     4       5       6
     7       8       9
>> A(i,j) % this is A([3 3 2 3],[1 2 3 3])
ans =
     7       8       9       9
     7       8       9       9
     4       5       6       6
     7       8       9       9
```

```
>> A(i,j) = 0 % this is A([3 3 2 3],[1 2 3 3]) also
A =
    1     2     3
    0     0     0
    0     0     0
```

从上面的代码可以看到，A(i,j)并没有执行用户希望的操作（相信倒数第二条语句中用户调用 A(i,j)的目的不是要得到结果中的数组，而是想查看满足条件的数组元素的值是什么。最后一条语句也可能与用户的初衷相悖）。可见，用户在编程时，想当然地将 A(k)与 A(i,j)视为同一概念是错误的。为了使读者弄清上例中倒数第二条语句的结果是如何得到的，我们将 A(i,j)做如下的等价扩展：

```
>> [A(3,1) A(3,2) A(3,3) A(3,3)
    A(3,1) A(3,2) A(3,3) A(3,3)
    A(2,1) A(2,2) A(2,3) A(2,3)
    A(3,1) A(3,2) A(3,3) A(3,3)]
ans =
    7     8     6     9
    7     8     6     9
    4     5     6     6
    7     8     6     9
```

可见，为了得到 A(i,j)，Matlab 首先用 i 中的各元素与 j 的第一个元素结合构成结果数组中第一列元素的索引值；然后用 i 中的各元素与 j 的第二个元素结合构成结果数组中第二列元素的索引值；依此类推。另外，从上边的结果可以看出，数组 A(i,j)的对角元素等于 A(k)的各元素。因此，下面的语句与 A(k)才是等价的：

```
>> diag(A(i,j))
ans =
    7
    8
    6
    9
```

虽然 diag(A(i,j))与 A(k)等价，但用户可能倾向于使用 A(k)，因为这种方法非常直观易懂，且具有较高的执行效率（因为该方法不用生成中间结果数组）。

根据前面对 A(i,j)所做的等价扩展，我们很容易理解 A(i,j)=0 等价于下面的语句：

```
>> A(3,1)=0; A(3,2)=0; A(3,3)=0; A(3,3)=0; % i(1) with all j
>> A(3,1)=0; A(3,2)=0; A(3,3)=0; A(3,3)=0; % i(2) with all j
>> A(2,1)=0; A(2,2)=0; A(2,3)=0; A(2,3)=0; % i(3) with all j
>> A(3,1)=0; A(3,2)=0; A(3,3)=0; A(3,3)=0; % i(4) with all j
A =
    1     2     3
    0     0     0
    0     0     0
```

由上面的分析可知，在用 find 函数进行满足某种关系的子数组搜索时，建议读者使用 A(k)，避免使用 A(i,j)。

当用户只需要在数组的某一部分查找满足条件的元素时，可以在 find 函数中添加适当的输入参数，避免其在整个数组中进行搜索。例如，下面的代码分别寻找 x 中大于 4 的所有元素、第一个元素、前两个元素以及最后两个元素：

```
>> x = randperm(8)  % new data
x =
    8    2    7    4    3    6    5    1
>> find(x>4)  % find all values greater than 4
ans =
    1    3    6    7
>> find(x>4,1) % find first value greater than 4
ans =
    1
>> find(x>4,1,'first') % same as above
ans =
    1
>> find(x>4,2) % find first two values greater than 4
ans =
    1    3
>> find(x>4,2,'last') % find last two values greater than 4
ans =
    6    7
```

上例又给出了 find 的几种调用格式：find(expr,n)、find(expr,n, 'first')和 find(expr,n, 'last')。在这些调用格式中，n 限制了 find 函数返回的最大索引个数，如果满足 expr 的元素个数小于 n，就返回满足 expr 的元素索引。

下表对 find 函数的各种用法进行了总结：

数组搜索语句	描述
i =find(X)	将数组 X 中不为 0（或为 True）的所有元素的单一索引值赋给 i
[r,c]=find(X)	将数组 X 中不为 0（或为 True）的所有元素的行索引值赋给 r，列索引值赋给 c
find(X,n) 或 find(X,n, 'first')	从数组 X 的第一个元素开始，寻找前 n 个不为 0（或为 True）的元素的索引值
find(X,n, 'last')	从数组 X 的最后一个元素开始，寻找最后 n 个不为 0（或为 True）的元素的索引值

用户除了需要知道数组中满足某一表达式的所有元素的索引值之外，有时候还需要知道数组中的最大值或最小值，以及它们出现的位置。Matlab 提供了函数 max 和 min 来满足用户的这一需要。例如，下面的代码分别求随机向量 v 中的最大值和最小值以及它们的索引位置：

```
>> v = rand(1,6) % new data
v =
    0.3046    0.1897    0.1934    0.6822    0.3028    0.5417
>> max(v) % return maximum value
```

```
ans =
    0.6822
>> [mx,i] = max(v) % maximum value and its index
mx =
    0.6822
i =
    4

>> min(v) % return minimum value
ans =
    0.1897

>> [mn,i] = min(v) % minimum value and its index
mn =
    0.1897
i =
    2
```

对于二维数组，max 和 min 将返回每一列的最大值或最小值，如下面的代码所示：

```
>> A = rand(4,6) % new data
A=
    0.1509    0.8537    0.8216    0.3420    0.7271    0.3704
    0.6979    0.5936    0.6449    0.2897    0.3093    0.7027
    0.3784    0.4966    0.8180    0.3412    0.8385    0.5466
    0.8600    0.8998    0.6602    0.5341    0.5681    0.4449
>> [mx,rx] = max(A)
mx =
    0.8600    0.8998    0.8216    0.5341    0.8385    0.7027
rx =
    4    4    1    4    3    2
>> [mn,rn] = min(A)
mn =
    0.1509    0.4966    0.6449    0.2897    0.3093    0.3704
rn =
    1    3    2    2    2    1
```

上例中，mx 和 mn 分别包含了数组 A 的每一列的最大值和最小值，rx 和 rn 则包含了是每一列的最大值和最小值在该列中的行索引。在很多情况下，用户并不想知道每一行的最大值或最小值，而希望知道整个数组的全局最大值或最小值。完成这一任务可以采用下面的两种方法：

```
>> mmx = max(mx) % first method: apply max again to prior result
 mmx =
    0.8998
>> [mmx,i] = max(A(:)) % second method: reshape A as a column vector first
mmx =
    0.8998
i =
    8
```

第一种方法是“最大值中选最大值”的方法，即 max(max(A))，它需要两次调用 max

函数。第二种方法则是先将二维数组延展成一个变量，然后再求最大值。相对而言，第二种方法要优于第一种，因为它仅调用一次 max 函数，并且还返回了全局最大值所在的单一索引值 i，即 mmx=A(i)。（第二种方法还可以用于多维数组，我们将在下一章讨论。）

当一个数组有多个相同的最大值或最小值时，函数 max 和 min 只能返回第一个出现的最大值或最小值的索引。例如，下例中 x 含有两个最大值 6，但 max 只能返回第一个 6：

```
>> x = [1 4 6 3 2 1 6]
 x =
     1     4     6     3     2     1     6
>> [mx ix] = max(x)
mx =
     6
ix =
     3
```

要想找出所有的最小值和最大值，需要借助函数 find，如下例所示：

```
>> i = find(x==mx) % indices of values equal to mx
i =
     3     7
```

（本书第 18 章将对 min 和 max 函数进行更详细的阐述。）

5.11 数组处理函数

Matlab 提供了一系列函数实现常见的数组处理功能。这些函数都比较容易掌握，读者可以边验证本节的例子，便学习这些函数的用法。

下面给出的几个例子介绍了数组结构变换函数的用法：

```
>> A = [1 2 3;4 5 6;7 8 9] % fresh data
 A =
     1     2     3
     4     5     6
     7     8     9
>> flipud(A) % flip array in up-down direction
ans =
     7     8     9
     4     5     6
     1     2     3
```

flipud 函数用于上下翻转数组。

```
>> fliplr(A) % flip array in the left-right direction
ans =
     3     2     1
     6     5     4
     9     8     7
```

flipud 函数则用于左右翻转数组。

```
>> rot90(A) % rotate array 90 degrees counterclockwise
```

```
ans =
     3     6     9
     2     5     8
     1     4     7
>> rot90(A,2) % rotate array 2*90 degrees counterclockwise
ans =
     9     8     7
     6     5     4
     3     2     1
```

rot90 用于将数组旋转 90 度的整数倍（如 90 度、180 度等）。

Matlab 7 还新增加了一个循环移行（列）函数 circshift，其使用方法如下面的例子所示：

```
>> A = [1 2 3;4 5 6;7 8 9] % recall data
 A =
     1     2     3
     4     5     6
     7     8     9
>> circshift(A,1)  % circularly shift rows down by 1
ans =
     7     8     9
     1     2     3
     4     5     6
```

circshift 按照第二个输入参数进行循环移行或移列，若该参数是一个标量，则 circshift 循环移行操作，次数是该标量指定的数值，当该值为正时，向下滚动循环，当该值为负时，向上滚动循环。

```
>> circshift(A,[0 1])  % circularly shift columns right by 1
ans =
     3     1     2
     6     4     5
     9     7     8
```

若第二个输入参数是一个两元素的向量，则 circshift 先按照第一个元素指定的值循环移行，然后再按照第二个元素指定的值循环移列。上面的代码由于第一个元素为 0，因此，circshift 仅执行循环移列操作。

```
>> circshift(A,[-1 1])  % shift rows up by 1 and columns right by 1
ans =
     6     4     5
     9     7     8
     3     1     2
```

该命令先执行向上滚动循环移行一次，再执行向右滚动循环移列一次。

reshape 函数是最通用的数组结构变换函数，下面给出了该函数的几个示例：

```
>> B = 1:12 % more data
B =
    1    2    3    4    5    6    7    8    9   10   11   12
>> reshape(B,2,6) % reshape to 2 row, 6 columns, fill by columns
```

```
ans =
    1     3     5     7     9    11
    2     4     6     8    10    12
>> reshape(B,[2 6]) % equivalent to above
ans =
    1     3     5     7     9    11
    2     4     6     8    10    12
>> reshape(B,3,4) % reshape to 3 rows, 4 columns, fill by columns
ans =
    1     4     7    10
    2     5     8    11
    3     6     9    12
>> reshape(B,3,[]) % MATLAB figures out how many columns are needed
ans =
    1     4     7    10
    2     5     8    11
    3     6     9    12
>> reshape(B,[],6)  % MATLAB figures out how many rows are needed
ans =
    1     3     5     7     9    11
    2     4     6     8    10    12
>> reshape(A,3,2) % A has more than 3*2 elements, OOPS!
??? Error using ==> reshape
To RESHAPE the number of elements must not change.
```

在使用 reshape 函数时，一定要保证变换前后数组中的元素个数不能改变，否则就会出错。

```
>> reshape(A,1,9) % stretch A into a row vector
ans =
    1     4     7     2     5     8     3     6     9
>> A(:)'  % convert to column and transpose; same as the above
ans =
    1     4     7     2     5     8     3     6     9
>> reshape(A,[],3) % MATLAB figures out how many rows are needed
ans =
    1     2     3
    4     5     6
    7     8     9
```

由上面的例子可以看出，reshape 函数既可以将二维数组变换成向量，又可以将向量变换为二维数组。

Matlab 还提供了一些函数用于将一个数组的部分元素提取出来生成另一个数组。

```
>> A % remember what A is
A =
    1     2     3
    4     5     6
    7     8     9
>> diag(A) % extract diagonal using diag
```

```
ans =
    1
    5
    9
>> diag(ans) % remember this ? same function, different action
ans =
    1       0       0
    0       5       0
    0       0       9
```

diag 函数既可以从一个方阵中提取对角元素组成一个向量，也可以将一个向量作为对角元素构成一个对角方阵，具体执行哪个操作，取决于给它提供的输入参数。

```
>> triu(A) % extract upper triangular part
ans =
    1       2       3
    0       5       6
    0       0       9
>> tril(A) % extract lower triangular part
ans =
    1       0       0
    4       5       0
    7       8       9
>> tril(A) - diag(diag(A)) % lower triangular part with no diagonal
ans =
    0       0       0
    4       0       0
    7       8       0
```

triu 函数保留方阵 A 的上三角部分（包括对角线），将下三角部分置为 0；tril 函数则保留方阵 A 的下三角部分（包括对角线），将上三角部分置为 0。

下边代码中给出的几个函数主要用于根据其他数组生成新的数组。

```
>> a =[1 2;3 4] % a smaller data array
a =
    1       2
    3       4
>> b = [0 1;-1 0] % another smaller data array
b =
     0      1
    -1      0
>> kron(a,b) % the Kronecker tensor product of a and b
ans =
     0      1       0      2
    -1      0      -2      0
     0      3       0      4
    -3      0      -4      0
```

kron 函数执行两个数组的 Kronecker 张量乘法。Kronecker 张量乘法对参数的顺序有要求，例如，上例中的 kron(a,b)等价于执行下面的操作：

```
>> [1*b 2*b
    3*b 4*b]
ans =
     0     1     0     2
    -1     0    -2     0
     0     3     0     4
    -3     0    -4     0
```

而 kron(b,a)执行的是不同的 Kronecker 张量乘法运算，如下面的代码所示：

```
>> kron(b,a) % the Kronecker tensor product of b and a
ans =
     0     0     1     2
     0     0     3     4
    -1    -2     0     0
    -3    -4     0     0
```

上例中的 kron(b,a)等价于执行下面的操作：

```
>> [0*a 1*a
    -1*a 0*a]
ans =
     0     0     1     2
     0     0     3     4
    -1    -2     0     0
    -3    -4     0     0
```

因此 kron(a,b)执行的操作是：将其第一个参数数组的每一个元素乘以第二个参数数组，然后形成一个分块矩阵。

最后，我们介绍一下数组复制函数 repmat，该函数我们在前边的章节中曾经介绍过。首先我们来看一个例子：

```
>> a % recall data
a =
    1    2
    3    4
>> repmat(a,1,3) % replicate a once down, 3 across
ans =
    1    2    1    2    1    2
    3    4    3    4    3    4
>> repmat(a,[1 3]) % equivalent to above
ans =
    1    2    1    2    1    2
    3    4    3    4    3    4
>> [a a a] % equivalent to above
ans =
    1    2    1    2    1    2
    3    4    3    4    3    4
>> repmat(a,2,2) % replicate a twice down, twice across
ans =
    1    2    1    2
    3    4    3    4
```

```
    1   2   1   2
    3   4   3   4
>> repmat(a,2) % same as repmat(a,2,2) and repmat(a,[2 2])
ans =
    1   2   1   2
    3   4   3   4
    1   2   1   2
    3   4   3   4
>> [a a; a a] % equivalent to above
ans =
    1   2   1   2
    3   4   3   4
    1   2   1   2
    3   4   3   4
```

函数 repmat 和 reshape 通常以两种方式接收第二个参数（索引参数）。第一种方式是接收用逗号分隔的参数，如 repmat(a,2,2)，有时也只接收一个标量参数，如 repmat(a,2)；第二种方式是接收一个行向量形式的参数，如 repmat(a,[1 3])。另外，对于 repmat 而言，如果第二个输入参数是一个单一的参数，如 repmat(A,n)，该命令将被解释为 repmat(A,[n,n])。

最后，要想从一个标量生成一个与某一数组具有相同维数的数组，用户可以使用类似于下面的命令：repmat(d,size(A))。例如，下面的代码生成一个与数组 A 大小相同的数组，其元素值均为 pi：

```
>> A = reshape(1:12,[3 4]) % new data
A =
    1   4   7   10
    2   5   8   11
    3   6   9   12
>> repmat(pi,size(A)) % pi replicated to be the size of A
ans =
    3.1416      3.1416      3.1416      3.1416
    3.1416      3.1416      3.1416      3.1416
    3.1416      3.1416      3.1416      3.1416
```

5.12 数组大小

Matlab 提供了函数 size、length 和 numel 来分别获取数组的行数和列数、数组长度（即行数或列数中的较大值）和元素总数。下例给出了 size 的用法：

```
>> A = [1 2 3 4;5 6 7 8]
A =
    1   2   3   4
    5   6   7   8
>> s = size(A)
s =
    2   4
```

当只有一个输出参数时，size 函数返回的是一个行向量，该行向量的第一个元素是数组的行数，第二个元素是数组的列数。

当有两个输出参数时，size 函数将数组的行数返回到第一个输出变量，将数组的列数返回到第二个输出变量，如下例所示：

```
>> [r,c] = size(A)
r =
    2
c =
    4
```

如果在 size 函数的输入参数中再添加一项，并用 1 或 2 为该项赋值，则 size 将返回数组的行数或列数，如下例所示：

```
>> r = size(A,1) % number of rows
r =
    2
>> c = size(A,2) % number of columns
c =
    4
```

由于一个数组 A(r,c)在声明时，第一个参数是其行数 r，因此 size(A,1)返回 A 的行数；第二个参数是其列数 c，因此 size(A,2)返回 A 的列数。

函数 numel 返回一个数组中元素的总数。例如，下面的代码求数组 A 中有多少元素：

```
>> numel(A)
ans=
    8
```

length 返回行数或列数的较大值。如下面的代码将返回 A 的列数：

```
>> length(A)
ans=
    4
```

对于向量，length 将返回该向量的长度，如下例所示：

```
>> B = -3:3
B =
    -3    -2    -1    0    1    2    3
>> length(B) % length of a row vector
ans =
     7
>> length(B') % length of a column vector
ans =
     7
```

另外，函数 size 和 length 同样适用于空数组（即 0 维数组）。请看下面的例子：

```
>> c = [] % you can create an empty variable!
c =
    []
>> size(c)
ans=
    0       0
>> d = zeros(3,0) % an array with one dimension nonzero!
```

```
d =
    Empty matrix:3-by-0
>> size(d)
ans =
    3       0
>> length(d)
ans =
    0
>> max(size(d)) % maximum of elements of size(d)
ans =
    3
```

size 和 length 在对空数组进行操作时，与普通的数组相似。只是有一点，对于空数组，无论空数组在其他维上是否为 0，length 函数只返回 0。

作为总结，下表给出了获取数组大小的一些语句和描述：

语句	描述
s=size(A)	该语句返回一个行向量 s，其第一个元素是 A 的行数，第二个元素是 A 的列数
[r,c]=size(A)	该语句返回两个标量 r 和 c，分别保存 A 的行数和列数
r=size(A,1)	该语句返回 A 的行数
c=size(A,2)	该语句返回 A 的列数
n=length(A)	如果 A 是非空数组，则相当于执行 max(size(A))；如果 A 为空数组，则返回 0；如果 A 是一个向量，则返回 A 的长度
n=max(size(A))	当 A 为非空数组时，返回 A 的最大维数；当 A 为空数组时，返回 A 中最长的非 0 维数
n=numel(A)	该语句返回数组 A 中的元素总数

5.13　数组和内存利用

随着现代计算机技术的发展，大多数数字处理器中都集成了浮点运算，因此，内存之间的数据存取时间已远远大于浮点计算的时间。由于内存存储速度比处理器处理速度慢，就使得一些计算机厂商不得不采用多级缓存措施，以便给处理器提供源源不断的数据。另外，当今计算机用户需要处理的数据量通常都很大，极易造成计算时的内存溢出。因此，掌握内存利用和管理方法对于编写高效的程序将起到至关重要的作用。

实际上，Matlab 本身并不参与任何内存管理，在 Matlab 中，内存分配和释放都需要调用标准 C 函数（如 malloc、calloc、free 等）。Matlab 主要靠编译器对这些库函数的执行，完成合理而高效的内存分配和释放。用户在编写和优化程序时，都需要在内存使用和执行速度之间实现折衷，在 Matlab 中，通常需要使用更多的内存来加快程序运行速度。正因为 Matlab 采用牺牲内存获得运行性能的策略，因此，掌握 Matlab 的内存分配原则，最大限度地减小内存溢出和内存碎片是一个熟练的 Matlab 用户必须具备的能力。

当用一个赋值语句生成一个变量时，例如下面的代码：

```
>> P = zeros(100);
```

Matlab 就会为变量 P 申请一个连续的内存区域进行存储，该内存区域的位置由编译器和操作系统决定。如果 P 被重新赋值，例如：

```
>> P =rand(5,6);
```

则 P 原来的内存区域将被释放，Matlab 为 P 重新申请一个相应大小的新的内存区域，新内存区域的位置仍由编译器和操作系统决定。很明显，当重新赋值的变量比原来的变量占用的空间大时，Matlab 就需要申请一个更大的连续内存区域来存储新数据。

即使重新赋值的变量和原来的变量所需的内存相同，Matlab 也会进行内存释放/再分配的操作。我们看下面的代码：

```
>> P = zeros(5,6); % same size as earlier
>> P = ones(6,5); % same number of elements as earlier
```

在上面的代码中，当第二条语句执行时，将把第一条语句分配给 P 的内存区域释放，然后重新为 P 申请一块相同大小的内存。Matlab 的这种内存分配方法，主要是为了最大限度地节省内存，例如，它可以随时消除那些已被赋值但不再使用的变量；但这种重复的释放和分配必然会加大系统开销，增加由于数据的来回存取消耗的时间。另外，上述 Matlab 内存分配方法只适用于向量和数组，对于标量而言，Matlab 只对其分配一次内存，当标量的值改变时，Matlab 只用新值代替变量原来所在内存位置中的值，而不再重新对其分配内存。例如，当给 a 赋值 a=2 后，再令 a=1，这时将只用 1 代替变量 a 原来的值 2，a 的存储位置不变。

根据 Matlab 对标量赋值的内存分配原则，当用户利用索引只对已经创建的数组中的某个元素进行赋值操作时，则 Matlab 只更新数组中该元素的值，不对该数组进行内存的释放/再分配操作。我们看下面的代码：

```
>> P(3,3) = 1;
```

上述赋值语句不会进行任何内存分配操作，因为上述语句本质上相当于对一个已经存在标量进行赋值。不过，上述语句是在不改变数组大小的情况下对数组中的一个元素进行赋值的，如果一条赋值语句使得一个数组的大小发生改变，则 Matlab 就会执行针对数组的内存释放/再分配操作。我们看下面的代码：

```
>> P(8,1) = 1;
>> size(P)
ans =
     8     5
```

上述语句使得 P 的行数增加，从一个 6×5 的数组增大为一个 8×5 的数组，这样，Matlab 就需要为这个新变量重新申请内存（该内存要比 P 原来的内存大），并将原来的变量拷贝进新的内存中，然后释放原来的内存区域。

如果可能的话，建议用户利用索引对数组中的每个元素进行赋值，而不要针对该数组进行赋值，这样可以消除由于内存分配/释放带来的系统开销。例如，下面的语句利用索引为 P 的每个元素赋值：

```
>> P(:) = ones(1,40);
```

在利用索引对数组中的每个元素进行赋值时，必须保证等号左边和右边的元素个数相同。对于上述语句，Matlab 仅仅将等号右侧的数据拷贝进等号左侧相应的数组元素的内存中，不存在内存的重新分配。该方法还可以保证赋值前后 P 的维数保持不变，如下面的代码结果所示：

```
>> size(P)
ans =
    8   5
```

有些用户在进行 Matlab 编程时，经常使用 global 命令将一个变量声明为全局变量。从内存管理的角度看，全局变量和普通的变量并无区别，因此，用户也无法通过全局变量的使用提高程序执行速度。

Matlab 自身也包含了一些提高程序运行性能的功能，其中一个重要的功能是延迟拷贝特性。为了理解该特性，我们先看下边的例子：

```
>> A = zeros(10);
>> B = zeros(10);
>> C = B;
```

在上面的例子中，Matlab 为变量 A 和 B 均分配了 10 个元素的内存，但并没有为变量 C 分配任何内存。变量 C 将共享分配给变量 B 的内存空间，直到 B 或者 C 发生了变化，才会为 C 分配内存，并将 B 中的数据拷贝到分配给 C 的内存区域中。因此，对于第三条语句，Matlab 并没有立即将 B 的数据拷贝到 C 中，而是延迟一段时间，直到 B 或者 C 发生了变化时才执行拷贝操作，这一特性就称为延迟拷贝特性。另外，如果程序将来仅仅引用 C 的数据，那么每次引用时将直接去访问 B 的相关内容，因此对于用户而言，C 实际上就是 B。只有当 B 的内容发生变化，或者 C 被赋以新值的时候，系统才花费时间为 C 分配内存，并将数组 B 拷贝进 C 新分配的内存区域中。对于小数组而言，延迟拷贝特性的优越性可能不甚明显，但对于很大的数组而言，该特性将会带来非常显著的程序运行性能的提高。

实际上，函数在引用其输入参数时，同样使用了延迟拷贝的方法。当我们调用一个函数时，例如，myfunc(a,b,c)，输入数组 a、b、c 并没有立即被拷贝进函数工作区，除非这些数组的值在某个地方被修改了。由于大多数函数都是仅仅从输入参数读取数据，并不修改输入参数的值，因此，采用延迟拷贝技术可以节省大量由于内存分配带来的系统开销。另外，由于采用了延迟拷贝技术，即使将一个很大的数组传递给函数，也不会对函数的运行性能产生太大影响。（该特性也会在本书第 12 章讲解 M 文件函数时涉及到。）

读者在编写 Matlab 程序时，有时会将一个函数的计算结果直接作为另一个函数的输入参数进行调用，下面就是一个例子：

```
>> prod(size(A))
ans =
   100
```

上例中，没有专门使用一个变量名存储 size(A)的计算结果，而是将其直接传递给了函数 prod。从内存管理的角度讲，上面的语句等价于：

```
>> tmp=size(A);
>> prod(tmp)
```

```
ans =
    100
```

从上面的等价语句可以看出，尽管前面的语句中并没有显式地生成一个变量来存储size(A)的结果，但 Matlab 在执行时仍生成了一个隐含的变量 tmp（为了阐述方便，作者任意取的名称），然后将 tmp 传递给函数 proc，计算完成后，tmp 被自动清除。因此，上述直接传递函数结果的方法的好处在于不用增加变量来存储中间结果，并且能自动将中间结果清除，避免了中间结果一直占用内存的情况。

在某些应用场合，用户需要在一个循环结构中不断增大一个数组的维数。这时比较好的方法是在不断增大维数之前，先为该数组分配一个足够大的内存。为了更好地理解上述内容，我们先看下面的例子：

```
>> A=1:5
A =
    1    2    3    4    5
>> B=6:10;
>> A=[A;B]
A = 1    2    3    4     5
    6    7    8    9    10
>> C=11:15;
>> A=[A;C]
A =
     1    2    3    4    5
     6    7    8    9   10
    11   12   13   14   15
```

在上边这个例子中，变量 A 每次被重新赋值时，Matlab 都为其分配新的内存，然后将原来的数据拷贝进新的内存，最后将原来的内存释放。如果涉及到的数组很大，或者反复赋值时，那么内存分配/释放带来的系统开销就会很大，程序执行的速度也会显著降低。因此，为了解决这一问题，建议用户预先为将要赋值的变量分配所需的内存，然后再根据需要填充这些内存。例如，我们可以采用下面的代码重新执行上面的操作：

```
>> A=zeros(3,5) % grab all required memory up front
A =
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0
>> A(1,:)=1:5;       % no memory allocation here
>> B=6:10;
>> A(2,:)=B;         % no memory allocation here
>> C=11:15;
>> A(3,:)=C          % no memory allocation here
A =
    1    2    3    4    5
    6    7    8    9    10
    11   12   13   14   15
```

尽管第一条语句看上去有点多此一举，但该例只为变量 A 分配了一次内存，在后续的语句中，都仅仅针对该内存中的值进行操作，没有涉及分配内存，显然应该比前面的代码执行效率高。

下表针对一些语法实例对本节内容进行了总结：

语法实例	描述
P=zeros(100); P=rand(5,6);	为一个数组变量重新赋值时，Matlab 会为该数组分配新的内存空间，并将原来的内存空间释放，从而带来由于内存分配和释放造成的系统开销
P(3,3)=1;	当利用索引为数组中的一个元素赋值时，Matlab 只会改变该元素内存中的值，不会对该数组进行内存分配/释放操作
P(8,1)=1;	当索引超出原数组的取址范围时，Matlab 会为该数组分配新的内存空间，并将原来的内存空间释放，从而带来由于内存分配和释放造成的系统开销
P=zeros(5,6);	即使不改变原数组的元素个数和数组结构，对数组进行重新赋值也会带来由于内存分配和释放造成的系统开销
P(:)=rand(1,30)	当利用索引为数组中每一个元素值进行赋值时，Matlab 仅仅将等号右侧的内容拷贝进等号左侧数组变量的内存区域中，不对左侧的数组变量进行内存分配/释放操作
B=zeros(10); C=B;	Matlab 在执行第二条语句时，采用延迟拷贝特性，即 C 和 B 共享分配给 B 的内存区域，直到 B 或 C 的值被改变
prod(size(A)) tmp=size(A); prod(tmp)	当将一个函数的调用结果直接传递给另一个函数时，Matlab 利用隐式变量进行内存分配。隐式变量和显式变量采用相同的方式分配内存。不过，为隐式变量分配的内存能够被自动清除
myfunc(a,b,c)	在函数读取输入参数中的数据时，也采用延迟拷贝特性。也就是说，只要函数仅仅引用变量 a、b、c 的值，而不改变它们的值，这些变量就不会被拷贝进函数工作区。因此，函数在读取输入参数的值时，不会带来由于内存分配和释放造成的系统开销
A=zeros(3,5); A(1,:)=1:5; A(2,:)=6:10; A(3,:)=11:15;	当需要不断增大一个变量的维数时，建议预先给该变量分配足够的内存。这样只会带来一次内存分配/释放的系统开销，而不是每次增大维数时都需要进行一次内存分配/释放操作

Chapter 6

多维数组

前一章，我们主要阐述了一维和二维数组的处理方法。从 Matlab 5 开始，Matlab 增加了对多维数组的支持。Matlab 对多维数组的处理方法（包括使用的函数和引用方式）大体上与一维和二维数组相同。目前，对于多维数组，只有第三维具有统一的名称，称为页（page），更高的维还不存在通用的叫法。因此，一个三维数组将由行、列和页三维组成，其中每一页包含一个由行和列构成的二维数组。上一章讲到，一个二维数组的所有列都必须包含相同数量的行，所有行也必须包含相同数量的列，同样，一个三维数组的所有页也必须包含相同数量的行和列。利用我们常见的电话号码薄可以很形象地描述三维数组的这一性质：一个电话号码薄（好比是一个三维数组）由许多页住宅电话列表（页）构成，每一页列表又都由相同数量的栏目（列）和联系名字（行）构成。

尽管数组的维数可以任意选取，但为了在验证过程中便于显示和观察，本章我们主要选择三维数组进行处理。

6.1 多维数组的创建

多维数组有多种创建方式。

首先，可以利用标准数组函数创建多维数组，如下例所示：

```
>> A = zeros(4,3,2)
A(:,:,1) =
     0     0     0
     0     0     0
     0     0     0
     0     0     0
A(:,:,2) =
     0     0     0
     0     0     0
     0     0     0
     0     0     0
```

上面的代码生成了一个 4 行、3 列、2 页的三维全 0 数组。从结果可以看出，该数组是按页显示的，首先显示第 1 页，然后显示第 2 页。除 zeros 外，ones、rand 和 randn 等函数

也可以按照相同的方法生成多维数组。

其次，可以利用直接索引方式生成多维数组，如下例所示：

```
>> A = zeros(2,3) % start with a 2-D array
A =
     0     0     0
     0     0     0
>> A(:,:,2) = ones(2,3) % add a second page to go 3-D!
A(:,:,1) =
     0     0     0
     0     0     0
A(:,:,2) =
     1     1     1
     1     1     1
>> A (:,:,3) = 4 % add a third page by scalar expansion
A(:,:,1) =
     0     0     0
     0     0     0
A(:,:,2) =
     1     1     1
     1     1     1
A(:,:,3) =
     4     4     4
     4     4     4
```

上边的代码首先生成了一个二维数组作为三维数组的第一页；然后，通过数组直接索引，添加第二和第三页。

再次，可以利用函数 reshape 和 repmat 生成多维数组。利用 reshape 函数生成多维数组的代码如下：

```
>> B = reshape(A,2,9) % 2-D data, stack pages side-by-side
B =
     0     0     0     1     1     1     4     4     4
     0     0     0     1     1     1     4     4     4

>> B = [A(:,:,1) A(:,:,2) A(:,:,3)] % equivalent to above
B =
     0     0     0     1     1     1     4     4     4
     0     0     0     1     1     1     4     4     4

>> reshape(B,2,3,3) % recreate A
ans(:,:,1) =
     0     0     0
     0     0     0
ans(:,:,2) =
     1     1     1
     1     1     1
ans(:,:,3) =
     4     4     4
     4     4     4

>> reshape(B,[2 3 3])% alternative to reshape(B,2,3,3)
ans(:,:,1)=
```

```
    0    0    0
    0    0    0
ans(:,:,2)=
    1    1    1
    1    1    1
ans(:,:,3)=
    4    4    4
    4    4    4
```

提示：reshape 函数可以将任何维数的数组转变成其他维数的数组。

利用 repmat 函数生成多维数组的代码如下：

```
>> C = ones(2,3) % new data
   C =
    1    1    1
    1    1    1

>> repmat(C,1,1,3) % this form not allowed above 2-D!
??? Error using ==> repmat
Too many input arguments.

>> repmat(C,[1 1 3]) % this is how to do it
ans(:,:,1) =
    1    1    1
    1    1    1
ans(:,:,2) =
    1    1    1
    1    1    1
ans(:,:,3) =
    1    1    1
    1    1    1
```

提示：repmat 是通过数组复制创建多维数组的。上边的代码即是将数组 C 在行维和列维上分别复制一次，然后在页维上复制三次得到了一个 2×3×3 的三维数组。

最后，可以利用 cat 函数创建多维数组，如下例所示：

```
>> a = zeros(2); % new data
>> b = ones(2);
>> c = repmat(2,2,2);
>> D = cat(3,a,b,c) % conCATenate a,b,c along the 3rd dimension
D(:,:,1) =
    0    0
    0    0
D(:,:,2) =
    1    1
    1    1
D(:,:,3) =
    2    2
    2    2

>> D = cat(4,a,b,c) % try the 4th dimension!
D(:,:,1,1) =
    0    0
```

```
    0    0
D(:,:,1,2) =
    1    1
    1    1
D(:,:,1,3) =
    2    2
    2    2

>> D(:,1,:,:) % look at elements in column 1
ans(:,:,1,1) =
    0
    0
ans(:,:,1,2) =
    1
    1
ans(:,:,1,3) =
    2
    2

>> size(D)
ans =
    2    2    1    3
```

从 size 函数的结果可以看出，上述代码生成的 D 是一个 2×2×1×3 的四维数组。

6.2 数组运算和处理

随着数组维数的增加，数组运算和处理就会变得越来越困难。对于多维数组而言，标量与数组之间的运算相对比较简单，并且与前一章介绍的二维数组大同小异，本节不再赘述。数组与数组之间的运算相对而言比较复杂，并且要求进行运算的两个数组在任何一维都必须具有相同的大小，本节将通过一些示例对其进行阐述。

Matlab 专门提供了一些函数对多维数组进行处理。squeeze 函数用于删除多维数组中的单一维（即大小为 1 的那些维），下例通过 squeeze 函数将四维数组 D 变为三维数组 E：

```
>> E = squeeze(D) % squeeze dimension 4 down to dimension 3
E(:,:,1) =
    0    0
    0    0
E(:,:,2) =
    1    1
    1    1
E(:,:,3) =
    2    2
    2    2
>> size(E)
ans =
    2    2    3
```

从上述代码可以看出，E 的数据和 D 一样，但比 D 少一维，只有 2 行、2 列和 3 页。

reshape 函数还可以将一个三维向量变为一维向量，如下例所示：

```
>> v(1,1,:) = 1:6 % a vector along the page dimension
v(:,:,1) =
     1
v(:,:,2) =
     2
v(:,:,3) =
     3
v(:,:,4) =
     4
v(:,:,5) =
     5
v(:,:,6) =
     6

>> squeeze(v) % squeeze it into a column vector
ans =
     1
     2
     3
     4
     5
     6

>> v(:) % this always creates a column vector
ans =
     1
     2
     3
     4
     5
     6
```

reshape 函数用于改变多维数组的行、列、页以及更高阶的维数，但不改变数组元素的总个数，如下例所示：

```
>> F =cat(3,2+zeros(2,4),ones(2,4),zeros(2,4)) % new 3-D array
F(:,:,1) =
     2     2     2     2
     2     2     2     2
F(:,:,2) =
     1     1     1     1
     1     1     1     1
F(:,:,3) =
     0     0     0     0
     0     0     0     0

>> G = reshape(F,[3 2 4]) % change it to 3 rows, 2 columns, 4 pages
G(:,:,1) =
     2     2
     2     2
     2     2
G(:,:,2) =
     2     1
     2     1
```

```
    1   1
G(:,:,3) =
    1   1
    1   0
    1   0
G(:,:,4) =
    0   0
    0   0
    0   0

>> H = reshape(F,[4 3 2]) % or 4 row, 3 columns, 2 pages
H(:,:,1) =
    2   2   1
    2   2   1
    2   2   1
    2   2   1
H(:,:,2) =
    1   0   0
    1   0   0
    1   0   0
    1   0   0

>> K = reshape(F,2,12) % 2 rows, 12 columns,1 page
K =
    2   2   2   2   1   1   1   1   0   0   0   0
    2   2   2   2   1   1   1   1   0   0   0   0
```

正确理解数组重组需要读者对多维空间有清晰的认识。另外，不同的数组重组具有不同的实际意义。例如，上例中的数组 G 其实用性就不如数组 K，因为 K 将 F 的页并排放在列的位置上，将数组由三维变为二维，显然更便于处理。

多维数组的重组方式与二维数组相同。重组时，数据将按照行→列→页→更高维的顺序进行整理。也就是说，从第一页开始，先整理该页第一列中所有的行数据，再整理第二列中所有的行数据，直到最后一列；当第一页整理完后，再以同样的方式整理第二页、第三页，直至最后一页。

函数 sub2ind 和 ind2sub 用于多维数组的直接引用，其索引顺序与前面讲的重组顺序相同，如下例所示：

```
>> sub2ind(size(F),1,1,1) % 1st row, 1st column, 1st page is element 1
ans =
    1
>> sub2ind(size(F),1,2,1) % 1st row,2nd column,1st page is element 3
ans =
    3
>> sub2ind(size(F),1,2,3) % 1st row,2nd column,3rd page is element 19
ans =
    19

>> [r,c,p]=ind2sub(size(F),19) % inverse of above
r =
    1
c =
    2
```

```
p =
    3
```

函数 flipdim 用于多维数组的翻转，相当于二维数组中的 flipud 和 fliplr 函数，例如，下面的代码分别对 M 进行按行、按列和按页的翻转：

```
>> M = reshape(1:18,2,3,3) % new data
M(:,:,1) =
    1   3   5
    2   4   6
M(:,:,2) =
    7    9  11
    8   10  12
M(:,:,3) =
    13  15  17
    14  16  18

>> flipdim(M,1) % flip row order
ans(:,:,1)
    2   4   6
    1   3   5
ans(:,:,2) =
    8   10  12
    7   9   11
ans(:,:,3) =
    14  16  18
    13  15  17

>> flipdim(M,2) % flip column order
ans(:,:,1) =
    5   3   1
    6   4   2
ans(:,:,2) =
    11   9  7
    12  10  8
ans(:,:,3) =
    17  15  13
    18  16  14

>> flipdim(M,3) % flip page order
ans(:,:,1)=
    13  15  17
    14  16  18
ans(:,:,2)=
    7   9   11
    8   10  12
ans(:,:,3)=
    1   3   5
    2   4   6
```

函数 shiftdim 用于循环轮换一个数组的维数。如果一个数组有 r 行、c 列和 p 页，则循环轮换一次，就生成一个 c 行、p 列和 r 页的数组，如下例所示：

```
>> M % recall data
M(:,:,1) =
     1     3     5
     2     4     6
M(:,:,2) =
     7     9    11
     8    10    12
M(:,:,3) =
    13    15    17
    14    16    18

>> shiftdim(M,1) % shift one dimension
ans(:,:,1) =
     1     7    13
     3     9    15
     5    11    17
ans(:,:,2) =
     2     8    14
     4    10    16
     6    12    18
```

从代码执行结果可以看出，将一个三维数组循环轮换一次，就使得其第一页的第一行变成了第一页的第一列，第一页的第二行变成了第二页的第一列，依此类推。下列代码将M循环轮换2次：

```
>> shiftdim(M,2) % shift two dimensions
ans(:,:,1) =
     1     2
     7     8
    13    14
ans(:,:,2) =
     3     4
     9    10
    15    16
ans(:,:,3) =
     5     6
    11    12
    17    18
```

从代码执行结果可以看出，将一个三维数组循环轮换两次，其第一页的第一列变成了第一页的第一行，第二页的第一列变成了第一页的第二行，第三页的第一列变成了第一页的第三行，依此类推。仅仅依靠上述数字结果很难使读者对循环轮换的概念形成直观的认识。因此，为了便于理解，我们将一个三维数组看作一个类似于魔方的方形盒子，从任何一个方向看，该盒子都由不同的层（页）组成，每一层都包括相同的行和列。这样，数组循环轮换就相当于旋转这个盒子使用户面对不同的盒面。

函数 shiftdim 也支持负的循环轮换次数。执行负循环轮换时，数组的维数将增加，并且多出的维数均为单一维。例如，下面的代码执行-1 次的循环轮换：

```
>> M % recall data
M(:,:,1) =
     1     3     5
```

```
    2   4   6
M(:,:,2) =
    7    9  11
    8   10  12
M(:,:,3) =
    13  15  17
    14  16  18
>> size(M) % M has 2 rows, 3 columns, and 3 pages
ans =
    2   3   3

>> shiftdim(M,-1) % shift dimensions out by 1
ans(:,:,1,1) =
    1   2
ans(:,:,2,1) =
    3   4
ans(:,:,3,1) =
    5   6
ans(:,:,1,2) =
    7   8
ans(:,:,2,2) =
    9   10
ans(:,:,3,2) =
    11  12
ans(:,:,1,3) =
    13  14
ans(:,:,2,3) =
    15  16
ans(:,:,3,3) =
    17  18
>> size(ans)
ans =
    1   2   3   3
```

由以上代码可知，三维数组 M 经-1 次循环轮换后变成了四维数组。请读者自己总结数组循环轮换的基本规律，找出数组轮换之前和之后的对应关系。

函数 permute 和 ipermute 用于实现多维条件下的转置操作。从本质上来说，permute 函数是 shiftdim 函数的扩展。考虑下边这个例子：

```
>> M % recall data
M(:,:,1) =
    1   3   5
    2   4   6
M(:,:,2) =
    7    9  11
    8   10  12
M(:,:,3) =
    13  15  17
    14  16  18

>> permute(M,[2 3 1] % same as shiftdim(M,1)
ans(:,:,1) =
    1    7  13
```

```
    3     9   15
    5    11   17
ans(:,:,2) =
    2     8   14
    4    10   16
    6    12   18

>> shiftdim(M,1)
ans(:,:,1) =
    1     7   13
    3     9   15
    5    11   17
ans(:,:,2) =
    2     8   14
    4    10   16
    6    12   18
```

在上面的代码中，permute 函数中的参数[2 3 1]表示使函数的第二维成为第一维，第三维成为第二维，第一维成为第三维。下面的代码则只在第一维和第二维之间进行转置：

```
>> permute(M,[2 1 3])
ans(:,:,1) =
     1        2
     3        4
     5        6
ans(:,:,2) =
     7        8
     9       10
    11       12
ans(:,:,3) =
    13       14
    15       16
    17       18
```

上述代码中的参数[2 1 3]指将数组 M 的行和列相互转置，页保持不变。转置完成后，结果数组中的每一页都是对原数组数据进行行列转置后得到的。

permute 函数的第一个参数为待转置的多维数组，第二个参数为转置顺序（ORDER），它必须是待转置的多维数组的维数的某种排列（如，对三维数组，必须是 1、2、3 这 3 个数的某种排列）；否则所要求进行的转置将无法进行。下边的例子将无法进行转置操作：

```
>> permute(M,[2 1 1])
??? Error using ==> permute
ORDER cannot contain repeated permutation indices.

>> permute(M,[2 1 4])
??? Error using ==> permute
ORDER contains an invalid permutation index.
```

函数 permute 还可以用来将一个数组变成一个更高维数的数组。例如，我们前面讲到的代码 shiftdim(M, -1)也可以用 permute 函数实现，代码为：

```
>> permute(M,[4 1 2 3])
ans(:,:,1,1) =
```

```
     1        2
ans(:,:,2,1) =
     3        4
ans(:,:,3,1) =
     5        6
ans(:,:,1,2) =
     7        8
ans(:,:,2,2) =
     9       10
ans(:,:,3,2) =
    11       12
ans(:,:,1,3) =
    13       14
ans(:,:,2,3) =
    15       16
ans(:,:,3,3) =
    17       18
```

上例中，我们对一个三维数组进行了四个维数的置换，这是因为，任何一个数组都具有大于其本身尺寸的更高维数，并且这些维数均为单位维数。例如，一个二维数组是具有页这一维的，并且仅有一页。总之，任何超过数组本身大小的维数都是单一维。对于上述代码而言，由于 M 是一个三维数组，其第四维必为单一维，因此，将 M 的第四维与第一维进行转置，第一维就变成了单一维。

对于一个二维数组而言，输入两次转置操作符就把数组变换回它原来的形式。对于多维数组，我们通常采用函数 ipermute 来取消 permute 所执行的转置操作。下例用 permute 转置后，又用 ipermute 将数组恢复为原来的形式：

```
>> M % recall data
M(:,:,1)=
     1     3     5
     2     4     6
M(:,:,2) =
     7     9    11
     8    10    12
M(:,:,3) =
    13    15    17
    14    16    18
>> permute(M,[3 2 1]) % sample permutation
ans(:,:,1) =
     1     3     5
     7     9    11
    13    15    17
ans(:,:,2) =
     2     4     6
     8    10    12
    14    16    18
>> ipermute(M,[3 2 1]) % back to original data
ans(:,:,1)=
     1     3     5
```

```
     2     4     6
ans(:,:,2) =
     7     9    11
     8    10    12
ans(:,:,3) =
    13    15    17
    14    16    18
```

6.3 数组大小

在 Matlab 中，计算数组大小的函数主要是 size 函数，该函数将返回数组每一维的大小。例如，下面的代码返回数组 M 的各维大小：

```
>> size(M) % return array of dimensions
ans =
    2   3   3
```

另外，函数 numel 用于返回数组的总元素个数，下面的代码返回数组 M 的元素个数：

```
>> numel(M) % number of elements
ans =
    18
```

当不指定 size 的返回值时，size 将返回一个由 M 的各维数组成的向量(如前面的例子)，当我们知道 M 的维数时，可以将 M 的维数返回到指定的变量中，如下例所示：

```
>> [r,c,p] = size(M) % return individual variables
r =
    2
c =
    3
p =
    3
>> r =size(M,1) % return just rows
ans =
    2
>> c = size(M,2) % return just columns
c =
    3
>> p = size(M,3) % return just pages
p =
    3
>> v = size(M,4) % default for all higher dimensions
v =
    1
```

我们不知道一个数组的维数或者某数组维数不确定时，可以利用函数 ndims 获得该数组的维数值，如下例所示：

```
>> ndims(M)
ans =
```

```
    3
>> ndims(M(:,:,1)) % just the 2-D first page of M
ans =
    2
```

在后一个例子中，因为 M(:,:,1)只有一页，实际上它是一个二维数组，所以 ndims 函数返回 2。函数 ndims 与下面的代码块等效：

```
>> length(size(M))
ans =
    3
```

下表总结了多维数组中常用的函数：

函数	描述
ones(r,c,…) zeros(r,c,…) rand(r,c,…) randn(r,c,…)	创建多维数组的基本函数，分别创建全 1、全 0、随机（介于 0~1 之间）和随机正态分布的多维数组
reshape(B,2,3,3) reshape(B,[2 3 3])	将一个数组变形为任意维数的数组
repmat(C,[1 1 3])	将一个数组复制成一个任意维数的数组
cat(3,a,b,c)	沿着一个指定的维将数组连接起来
squeeze(D)	删除大小等于 1 的维，也就是单一维
sub2ind(size(F),1,1,1) [r,c,p]=ind2sub(size(F),19)	将下标转化为单一索引值，或将单一索引值转化为下标
flipdim(M,1)	沿着一个指定的维轮换顺序。等效于二维数组中的 flipud 和 fliplr
shiftdim(M,2)	循环轮换。第二个参数为正的情况下，进行各维的循环轮换；第二个参数为负的情况下，将使数组的维数增加
permute(M,[2 1 3]) ipermute(M,[2 1 3])	多维数组的转置操作，其中前者为转置操作，后者为取消转置操作
size(M) [r,c,p]=size(M)	返回数组各维的大小
r=size(M,1)	返回数组的行数
c=size(M,2)	返回数组的列数
p=size(M,3)	返回数组的页数
ndims(M)	获取数组的维数
numel(M)	获取数组的元素总个数

Chapter 7

数字数据类型

在前面的章节中，本书所提到的数字变量都是以双精度格式保存的实数或复数数组。双精度格式是 Matlab 7 之前版本中所有数据的存储格式。稍有编程经验的用户都知道，字符串中的一个字符至多需要 2 字节的存储空间，而逻辑数组中的一个逻辑元素（True 或 False）则仅需要 1 个比特就足够了。因此，将字符串和逻辑数组都存储为 8 字节的双精度数组，必然造成极大的存储空间的浪费。

随着 Matlab 7 版本的不断升级，数据的存储效率逐渐提高。先是字符串成为了一个独立的数据类型（又称为变量类），变为每字符由 2 字节表示；之后，逻辑数组也成为了一个独立的数据类型，变为每逻辑元素由 1 字节表示；最近，Matlab 7 又新增了单精度数据类型和各种有符号和无符号整数数据类型（见 7.1 节表格）。

Matlab 7 之前的版本都不支持单精度和整数数据类型的算术运算，但支持这两种数据类型的存储、查询、逻辑比较以及数组处理。因此，在 Matlab 7 之前的版本中，要想执行这些数据类型的算术运算，就必须首先进行数据转换，将其变为双精度格式。之后，可以根据需要将计算结果转换为原来的数据类型。随着 Matlab 7 版本的发布，Matlab 本身已能够进行这些数据类型的大部分运算，无需用户进行专门的数据转换。

7.1 整数数据类型

Matlab 7 支持 8 位、16 位、32 位和 64 位的有符号和无符号整数数据类型。下表对这些数据类型进行了总结：

数据类型	描述
uint8	8 位无符号整数，范围是 0~255（或 $0\sim2^8$）
int8	8 位有符号整数，范围是-128~127（或$-2^7\sim2^7-1$）
uint16	16 位无符号整数，范围是 0~65 535（或 $0\sim2^{16}$）
int16	16 位有符号整数，范围是-32 768~32 767（或$-2^{15}\sim2^{15}-1$）

（续表）

数据类型	描述
uint32	32 位无符号整数，范围是 0~4 294 967 295（或 0~2^{32}）
int32	32 位有符号整数，范围是-2 147 483 648~2 147 483 647（或-2^{31}~2^{31}-1）
uint64	64 位无符号整数，范围是 0~18 446 744 073 707 551 615（或 0~2^{64}）
int64	64 位有符号整数，范围是-9 223 372 036 854 775 808~9 223 372 036 854 775 807（或-2^{63}~2^{63}-1）

上表中的整数数据类型除了定义范围不同外，具有相同的性质。其定义范围可以通过函数 intmax 和 intmin 获得，其中 intmax 获得范围的上限，intmin 获得范围的下限。下例分别获取 int8 和 uint32 的上限和下限：

```
>> intmax('int8')
ans =
   127
>> intmin('uint32')
ans =
   0
```

我们可以通过多种方式创建一个整数数据变量。当我们需要一个全 0 或全 1 数组时，可以使用函数 zeros 和 ones，下例分别创建 int8 类型的全 0 数组和 uint16 类型的全 1 数组：

```
>> m = zeros(1,6,'int8') % specify data type as last argument
m =
    0    0    0    0    0    0
>> class(m) % confirm class of result
ans =
int8
>> n = ones(4,'uint16') % again specify data type as last argument
n =
    1    1    1    1
    1    1    1    1
    1    1    1    1
    1    1    1    1
>> class(n) % confirm class of result
ans =
uint16
```

当我们需要非全 0 或全 1 数组时，就需要将计算结果转换为所需的数据类型，下例用两种转换方法创建了一个 uint8 类型的整数数组：

```
>> k = 1:7 % create default double precision
k =
    1    2    3    4    5    6    7
>> class(k)
ans =
double
>> kk = uint8(k) % convert using uint8 function
kk =
    1    2    3    4    5    6    7
```

```
>> class(kk)
ans =
uint8
>> kkk = cast(k,'uint8') % use more general cast function
kkk =
    1    2    3    4    5    6    7
>> class(kkk)
ans =
uint8
```

创建后的变量不会因为其他类型数据的插入而改变自身的数据类型。下面的例子就表明了这一点：

```
>> kkk(3:5) = ones(1,3) % insert double precision values
kkk =
    1    2    1    1    1    6    7
>> class(kkk) % class remains unchanged
ans =
uint8
>> kkk(5:7) = zeros(1,3,'uint16') % insert uint16 data
kkk =
    1    2    1    1    0    0    0
>> class(kkk) % class remains unchanged
ans =
uint8
>> kkk(1:2:end) = pi % insert a noninteger value!
kkk =
    3    2    3    1    3    0    3
>> class(kkk) % class remains unchanged
ans =
uint8
```

注意：当插入一个非整数（如前例中的 pi）时，程序首先将其进行四舍五入取整，然后再插入。

下面的代码演示了同类整数数据类型之间的数学运算：

```
>> k = int8(1:7) % create new data
k =
    1    2    3    4    5    6    7
>> m = int8(randperm(7)) % more new data
m =
    7    2    3    6    4    1    5

>> k+m % addition
ans =
    8    4    6    10   9    7    12
>> k-m % subtraction
ans =
    -6   0    0    -2   1    5    2
>> k.*m % element by element multiplication
ans =
    7    4    9    24   20   6    35
```

```
>> k./m % element by element division
ans =
     0     1     1     1     1     6     1

>> k % recall data
k =
     1     2     3     4     5     6     7

>> k/k(2)
ans =
     1     1     2     2     3     3     4
```

在这些代码中，加法、减法和乘法都比较容易理解，除法稍微复杂一些，因为多数情况下，整数的除法并不一定得到整数结果。Matlab 在进行整数除法时，先将数组视为双精度类型进行除法运算，然后再对结果进行四舍五入取整。

> Matlab 在进行整数除法时，首先基于双精度类型进行运算，然后将运算结果进行四舍五入取整，最后把取整结果转换为原来的整数数据类型。

Matlab 至今还未定义不同整数数据类型之间的数学运算，不过，一个双精度类型的标量和一个整数数据类型的数组之间可以进行数学运算，此时，Matlab 先把双精度类型标量转化为相应的整数数据类型，然后执行需要的数学运算，如下例所示：

```
>> m % recall data and data type
m =
     7     2     3     6     4     1     5
>> class(m)
ans =
int8
>> n = cast(k,'uint16') % new data of type uint16
n =
     1     2     3     4     5     6     7
>> m+n % try mixed type addition
??? Error using ==> plus
Integers can only be combined with integers of the same class, or scalar
doubles.
>> n+3 % try adding default double precision constant 3
ans =
     4     5     6     7     8     9    10
>> class(ans)
ans =
uint16
>> n-(1:7) % try nonscalar double precision subtraction
??? Error using ==> minus
Integers can only be combined with integers of the same class, or scalar
doubles.
```

> Matlab 仅支持一个双精度类型的标量和一个整数数据类型数组之间的混合数学运算，但不支持一个双精度类型的数组和一个整数数据类型数组之间的混合数学运算。

由于每种整数数据类型都有相应的取值范围，因此数学运算有可能产生结果溢出（即得出超过相应整数数据类型取值范围的结果）。Matlab 利用饱和处理处理此类问题，即当运算结果超出了由函数 intmin 和 intmax 指定的上下限时，就将该结果设置为 intmin 或 intmax 的返回值，到底是哪一个，主要看溢出的方向。具体见下例：

```
>> k = cast('hellothere','uint8') % convert a string to uint8
k =
   104   101   108   108   111   116   104   101   114   101
>> double(k)+150 % perform addition in double precision
ans =
   254   251   258   258   261   266   254   251   264   251
>> k+150 % perform addition in uint8, saturate at intmax('uint8')=255
ans =
   254   251   255   255   255   255   254   251   255   251
>> k-110 % perform subtraction in uint8, saturation at intmin('uint8')=0
ans =
   0   0   0   0   1   6   0   0   4   0
```

总之，Matlab 支持各种整数数据类型。除了 64 位整数数据类型外，其他整数数据类型都具有比双精度类型较高的存储效率。基于同一整数数据类型的数学运算将产生相同数据类型的结果。混合数据类型的运算仅限于在一个双精度标量和一个整数数据类型数组之间进行。有一点我们在前面的例子中没有演示，就是整数数据类型中不存在双精度数据类型中常见的 inf 和 NaN。另外，Matlab7 中没有定义基于 64 位整数数据类型的数学运算，估计在以后的版本中应该会涉及到。

7.2　浮点数据类型

前面多次讲到，Matlab 的默认数据类型是双精度类型，简写为双精度（double）。本节的浮点数据类型同样符合 IEEE 为双精度数据类型运算制定的标准。为了节省存储空间，Matlab 也支持单精度数据类型的数组。基于单精度数据类型的数学运算的定义和操作与上节整数数据类型基本相同。在下面的例子中，我们分别用函数 realmin、realmax 和 eps 得到单精度数据类型的取值范围和精度。

```
>> realmin('single')
ans =
   1.175494350822288e-038
>> realmax('single')
ans =
   3.402823466385289e+038
>> eps('single')
ans =
   1.192092895507813e-007

>> realmin('double') % compare to corresponding double values
ans =
   2.225073858507201e-308
>> realmax('double')
```

```
ans =
   1.797693134862316e+308
>> eps % same as eps(1) and eps('double')
ans =
   2.220446049250313e-016
```

单精度数据的创建与前述整数数据的创建方法相同，如下例：

```
>> a = zeros(1,5,'single') % specify data type as last argument
a =
   0    0    0    0    0
>>  b = eye(3,'single') % specify data type as last argument
b =
   1    0    0
   0    1    0
   0    0    1
>> c = single(1:7) % convert default double precision to single
c =
   1    2    3    4    5    6    7
>> d = cast(6:-1:0,'single') % use more general cast function
d =
   6     5     4     3     2     1    0
```

单精度数据之间或单精度与双精度数据之间的数学运算的结果将为单精度数，如下例：

```
>> c.^d % element by element exponentiation of singles
ans =
   1    32    81    64    25    6    1
>> c*pi % multiplication by a scalar double
ans =
   3.1416    6.2832    9.4248    12.566    15.708    18.85    21.991
>> d.*rand(size(d)) % element by element multiplication by a double array
ans =
   4.879   0.049307    0.55556    0.6083    0.39744    0.60379    0
>> class(ans)
ans =
single
```

Matlab 支持双精度和单精度数组之间的数学运算，返回单精度值。

单精度数据类型中包含双精度数据类型中常见的特殊浮点值 inf 和 NaN，如下例：

```
>> c % recall data
c =
   1    2    3    4    5    6    7
>> c(1:2:end) = 0 % inserting double precision does not change data type
c =
   0    2    0    4    0    6    0
>> c./c % create 0/0 values
Warning: Divide by zero.
Ans =
   NaN    1    NaN    1    NaN    1    NaN
```

```
>> 1./c % create 1/0 values
Warning: Divide by zero.
Ans =
   Inf   0.5   Inf   0.25   Inf   0.16667   Inf
```

7.3　小结

下表给出了适合 Matlab 7 支持的数字数据类型的函数。

函数	描述
double	创建或转化为双精度数据类型
single	创建或转化为单精度数据类型
int8, int16, int32, int64	创建或转化为有符号整数数据类型
uint8, uint16, uint32, uint64	创建或转化为无符号整数数据类型
isnumeric	若是整数或浮点数据类型，返回 True
isinteger	若是整数数据类型，返回 True
isfloat	若是单精度或双精度数据类型，返回 True
isa(x, 'type')	'type'包括'numeric'、'integer'和'float'（下同），当 x 类型为 type 时，返回 True
cast(x, 'type')	将 x 类型置为'type'
Intmax('type')	'type'数据类型的最大整数值
intmin('type')	'type'数据类型的最小整数值
realmax('type')	'type'数据类型的最大浮点实数值
realmin('type')	'type'数据类型的最小浮点实数值
eps('type')	'type'数据类型的 eps 值（浮点值）
eps(x)	x 的 eps 值：即 x 与 Matlab 能表示的和其相邻的同数据类型的那个数之间的距离
zeros(...,'type')	创建数据类型为'type'的全 0 阵列
ones(...,'type')	创建数据类型为'type'的全 1 阵列
eye(...,'type')	创建数据类型为'type'的单位阵列

Chapter 8

单元数组和结构体

Matlab 5 增加了两种新的承载数据的类型：单元数组（cell array）和结构体（structure）。这两种数据类型均是将类型不同的相关数据集成到一个单一的变量中，这使得大量的相关数据的处理与引用变得简单而方便。需要注意的是，单元数组和结构体仅仅是承载其他数据类型的容器，大部分数学运算只针对其中的具体数据进行，而不针对单元数组和结构体本身进行。因此，在进行数学运算之前，用户必须确认一个单元数组或者结构体内含有特定的数据变量。

为了更形象地理解单元数组，我们将其比作小区墙上一排排的邮箱来说明。这些邮箱的集合即构成了一个单元数组，其中每一个邮箱就是这个数组中的一个单元。每个邮箱中所存放的物品是不一样的，这相当于每一个单元中所包含的数据类型或数组维数是不同的，例如一个字符串或者一个多维数组等。每个邮箱都是用一个数字（如 001）来标识其地址的，同样，单元数组中的每一个单元也是用一个数字来索引的。当我们向一个邮箱发送邮件的时候，需要明确指出该邮箱的号码，同样，当用户将数据加入到一个单元中或者从某个单元中提取数据时，也需要标识出该单元的索引号。

结构体与单元数组十分相似，惟一的不同之处在于结构体中的数据存储不是由数字来标识的，而是由名称来标识的。我们仍旧用邮箱来打比方，邮箱的集合就是结构体，每个邮箱相当于结构体中的一个结构元素，只不过这时每个邮箱都是用其所有者的名字（如张三）标识的，而不是用数字代码来标识。当我们向一个邮箱发送信件时，需要明确指出接收信件的邮箱的名字（如张三），同样，当用户将数据添加到一个特殊的结构元素中，或从一个结构元素中提取数据时，也需要标识出该结构元素的名称（即域名）。

8.1 单元数组的创建

单元数组中的每一个元素称为单元（cell）。单元可以包含任何类型的 Matlab 数据，这些数据类型包括数值数组、字符、符号对象，甚至其他的单元数组和结构体。不同的单元可以包含不同的数据，例如，在一个单元数组中，一个单元可以包含一个数值数组，另一个单元可以包含一个字符串数组，而第三个单元可以包含一个复数向量。从原理上讲，Matlab 可以创建任意维数的单元数组，但在大多数情况下，为了处理方便，用户通常只需

要创建一个简单的单元向量（即一维的单元数组）即可。

用户可以通过两种方式创建一个单元数组：一是通过赋值语句直接创建；二是利用 cell 函数先为单元数组分配一个内存空间，然后再给各个单元赋值。在后面的例子中，我们将分别利用这两种方法创建单元数组。有一点请读者注意，如果你在验证这些例子时，无法得到与书中相同的结果（有时可能出现运行错误），那么很有可能是因为你所创建的单元数组与工作区中某一变量重名了。请用户将工作区中与单元数组重名的那个变量删除，或者将单元数组命名为与工作区中的各个变量均不同的名字，然后重试，即可得出预期的结果。

> 当用户试图将一个单元赋值给一个非单元数组类型的变量（如一个字符串变量）时，Matlab 就会停止运行并报错。

下面，我们来看一下单元数组的第一种创建方式：直接赋值法。该方法通过给每个单元逐个赋值来创建单元数组，根据在赋值时对单元访问方式的不同，又有两种赋值方法。一种方法采用标准数组索引法进行赋值，赋值时必须将赋给单元的值用花括号（即{}）括起来，这个花括号表明其中的表达式是单元中的内容，而不是一个普通的数组或字符串，我们称这种方法为“按单元索引法”。下例创建了一个 2×2 的单元数组，并利用“按单元索引法”对每一个单元进行直接赋值：

```
>> clear A % make sure A is not being used
>> A(1,1) = { [1 2 3; 4 5 6; 7 8 9]};
>> A(1,2) = { 2+3i }; % semicolons suppress display
>> A(2,1) = { 'A character string'};
>> A(2,2) = { 12:-2:0 } % no semicolon, so display requested
A =
            [3×3 double]        [2.0000+3.0000i]
     'A character string'          [1×7 double]
```

从上例我们可以看到，Matlab 并没有将单元数组 A 中所有的单元内容显示出来，这主要是因为 A 中有些单元（例如 A(1,1)和 A(2,2)）占有比较大的显示空间，Matlab 为了显示方便，只显示了这些内容的大小和数据类型。

另外，还有一种方法称为“按内容索引法”，该方法将花括号写在了等号左边，右边是将要赋的值。例如，下面代码利用“按内容索引法”生成了与前面相同的单元数组 A：

```
>> A{1,1} = [1 2 3; 4 5 6; 7 8 9];
>> A{1,2} = 2+3i;
>> A{2,1} = 'A character string';
>> A{2,2} = 12:-2:0;
A =
            [3×3 double]        [2.0000+3.0000i]
     'A character string'          [1×7 double]
```

上例中，语句 A{1,1} = [1 2 3; 4 5 6; 7 8 9]表明 A 是一个单元数组，并且 A 的第一行第一列的值等于[1 2 3; 4 5 6; 7 8 9]。

注意：“按单元索引法”和“按内容索引法”是完全等效的，它们可以互换使用。

通过上面两个例子，我们可以知道，花括号{}用于访问单元的值，而圆括号( )用于标

识单元（不用于访问单元的值）。我们仍以邮箱为例，假如 02 号邮箱中有一个包裹，则花括号用于查看包裹，而圆括号则用于标识该邮箱为 02 号邮箱。

> A(i, j)={x}和 A{i, j}=x 都表明将 x 的值保存在单元数组 A 的第（i, j）个元素中。其中，前者 A(i, j)={x}称为"按单元索引"；后者 A{i, j}=x 称为"按内容索引"。换句话说，花括号{ }用于访问单元的值，而圆括号( )用于标识单元而不查看它们的值。

在上面两个例子中，为了显示方便，A 并没有将所有的单元内容完全显示出来。要想完全显示所有单元的内容，我们可以使用 celldisp 函数。下面的代码将单元数组 A 的所有单元都显示出来：

```
>> celldisp(A)
A{1,1} =
    1   2   3
    4   5   6
    7   8   9
A{2,1} =
    A character string
A{1,2} =
    2.0000 + 3.0000i
A{2,2} =
    12  10  8   6   4   2   0
```

celldisp 函数属于强制显示指令，不论单元数组有多少单元，也不论每个单元有多少内容，都将在命令窗口中输出显示出来。很显然，当一个单元数组单元较多，或者某一单元的内容较长时，会使 Matlab 的命令窗口中充满了显示的内容，这通常是用户不愿意看到的。在这种情况下，我们可以利用"按内容索引"或"按单元索引"的方式，只显示出某一个单元的值。下面的代码分别演示了两种索引显示的方法：

```
>> A{2,2} % content addressing
ans =
    12      10      8       6       4       2       0
>> A(2,2) % cell indexing
ans =
    [1x7 double]
>> A{1,:} % address contents of the first row
ans =
    1       2       3
    4       5       6
    7       8       9
ans =
    2.0000 + 3.0000i
>> A(1,:)
ans =
    [3×3 double]   [2.0000+ 3.0000i]
```

从上面的结果可以发现，"按内容索引"和"按单元索引"所显示的单元内容是不同的，

“按内容索引”能够显示完整的单元内容，而“按单元索引”有时无法显示完整的单元内容（请读者分析其中的原因）。另外，由于我们只对单元数组进行了命名（即 A），没有对其中的每个单元进行命名，因此，上例进行显示时，所有的单元值都被命名为 ans。

当我们创建的单元数组比较简单时，可以采用更为简单的方法。如下面的例子仅用一对花括号，用逗号分隔列，用分号分隔行，创建了一个 2×2 的单元数组：

```
>> B = {[1 2], 'John Smith'; 2+3i, 5}
B=
           [1×2 double]     'John Smith'
     [2.0000 + 3.0000i]     [         5]
```

前面阐述了单元数组的第一种创建方法：直接赋值法。单元数组还有另外一种创建方法：cell 函数法。即，首先利用 cell 函数生成一个空的单元数组，然后再向其中添加所需的数据。下面的代码生成了一个 2×3 的空单元数组：

```
>> C = cell(2,3)
C =
    []      []      []
    []      []      []
```

利用 cell 函数生成空单元数组后，可以采用“按内容索引”和“按单元索引”方法对其进行赋值。在赋值时，用户一定要注意花括号和圆括号的正确用法，如下面的代码即是括号使用不当造成出错：

```
>> C(1,1) = 'This doesn''t work'
??? Conversion to cell from char is not possible.
```

上例中，等号左边使用了“按单元索引”，因此等号右边必须是一个单元，而实际上等号右边并不是一个单元，因为单元的值必须用花括号括起来才行。所以，Matlab 最终就发出一条出错信息。下例给出了正确的赋值方法：

```
>> C(1,1) = { 'This does work' }
C =
    'This does work'    []      []
                  []    []      []
>> C{2,3} = 'This works too'
C =
    'This does work'    []                    []
                  []    []    'This works too'
```

上例的第二条语句使用了“按内容索引”，所以 Matlab 就把等号右侧的字符串视为左侧单元的值直接赋给该单元。

8.2　单元数组的处理

从某种程度上讲，单元数组是数值数组的扩展，只不过其包含的内容由单一类型扩展到多种类型，因此，单元数组的处理方法与前几章讲到的数值数组的处理方法基本相同。例如，当用户给单元数组赋值时指定了超出数组大小的索引号（如给 8.1 节的 C(3,4)赋值）

时，Matlab 就自动将这个数组进行扩展，并且用空数组[ ]将新增加的单元填满。

用户可以利用方括号将多个单元数组组合在一起，形成一个维数更大的单元数组，例如，下面的代码将 A、B 沿列的方向组合在一起，形成了 C：

```
>> A % recall prior cell arrays
A =
            [3×3 double]    [2.0000 + 3.0000i]
   'A character string'           [1×7 double]
>> B
B =
            [1×2 double]          'John Smith'
     [2.0000 + 3.0000i]          [          5]
>> C=[A;B]
C =
            [3×3 double]    [2.0000 + 3.0000i]
   'A character string'           [1×7 double]
            [1×2 double]          'John Smith'
     [2.0000 + 3.0000i]    [                 5]
```

想要获得一个单元数组中的子数组时，可以利用传统的数组索引方法，将一个数组的子集提取出来放到一个新的单元数组中。例如，下例抽取 C 的第一行和第三行组成 D：

```
>> D = C([1 3],:) % first and third rows
D =
    [3×3 double]          [2.0000 + 3.0000i]
    [1×2 double]          'John Smith'
```

另外，将单元数组的某一部分设置为空数组，可以删除该部分的内容。例如，下例将 C 的第三行删除：

```
>> C(3,:) = []
C =
            [3×3 double]          [2.0000 + 3.0000i]
    'A character string'                [1×7 double]
     [2.0000 + 3.0000i]          [                 5]
```

注意：由于在前面的一系列处理过程中，我们只对数组单元本身进行处理，而不涉及到单元的值，因此，我们只用到了“按单元索引”，即圆括号索引方式，没有用到花括号。

在单元数组中，我们同样可以利用函数 reshape 改变一个单元数组的结构。不过，该函数不能用来添加或删除单元，也就是说，单元数组改变前后的总单元数保持不变。下例利用 cell 生成了一个 3×4 的单元数组 X，并利用函数 reshape 将其变为 6×2 的单元数组 Y(总单元数 12 保持不变)：

```
>> X = cell(3,4);
>> size(x) % size of the cell array, not the contents
ans =
     3    4

>> Y = reshape(x,6,2);
>> size(Y)
ans =
     6    2
```

上例中，函数 size 返回单元数组的大小，用于验证单元数组的变化。实际上，当总单元数保持不变时，函数 reshape 可以将一个单元数组变成任何结构，而不用考虑各单元的类型。

从名字上看，repmat 函数似乎是专门为矩阵设立的函数，其实，它也同样适用于单元数组（包括后面讲到的结构体），表示复制单元数组。下面的代码利用 repmat 函数将 Y 复制三次构成 Z:

```
>> Y % recall data
Y =
    []        []
    []        []
    []        []
    []        []
    []        []
    []        []
>> Z = repmat(Y,1,3)
Z =
    []        []        []        []        []        []
    []        []        []        []        []        []
    []        []        []        []        []        []
    []        []        []        []        []        []
    []        []        []        []        []        []
    []        []        []        []        []        []
```

8.3　单元内容的获取

为了获取单元数组中一个单元的值，用户必须使用“按内容索引”，即使用花括号进行索引。例如，下例将单元数组 B 的第二行、第二列的值赋给 x:

```
>> B % recall cell array
B =
          [1×2 double]          'John Smith'
    [2.0000 + 3.0000i]          [         5]
>> x = B{2,2} % content addressing uses {}
x =
    5
```

由于 B 的第二行、第二列的值为一个标量值 5，因此 x 也应该是一个标量。我们可以用函数 class 查看该标量的类型，如下例所示：

```
>> class(x) % return argument's data type
ans =
     double
```

通过 class 的返回值可以看出，x 是一个 double 型的标量。

也许有的读者会问，使用“按单元索引”就不能获取单元的值吗？为了回答这个问题，我们来看下面的例子：

```
>> y = B(2,2)        % cell indexing uses( )
y =
    [5]
>> y = B(4)          % same as above using single index
y =
    [5]

>> class(y)          % y is not a double, but a cell!
ans =
cell
>> class(y{1})       % but the contents of y is a double!
ans =
double
```

从上面的例子可以看出，“按单元索引”与“按内容索引”得到的结果是完全不同的。“按内容索引”获得的是单元的值，而“按单元索引”获得的是单元标识，从上例第三条语句（class(y)）可清楚看到这一点。

除了可以用前面的 class 函数来检测单元数组各单元的数据类型外，Matlab 还提供了其他一些函数来判断数组单元的类型，如 iscell、isa、isnumeric 等，这些函数均为逻辑判断函数，将返回逻辑结果（即 True(1)或 False(0)）。下面的代码给出了这些函数的具体应用：

```
>> iscell(y)             % yes, y is a cell
ans =
    1
>> iscell(y{1})          % contents of y is NOT a cell
ans =
    0
>> isa(y,'cell')         % yes, y is  a cell
ans =
    1
>> isdouble(y{1})        % this function doesn't exist (yet?)
??? Undefined function or variable 'isdouble'
>> isnumeric(y{1})       % contents of y is numerical
ans =
    1
>> isfloat(y{1})         % contents of y is floating point
ans =
    1

>> isa(y{1},'double')    % contents of y is a double
ans =
    1
>> isa(y{1},'numeric')   % contents of y is also numeric
ans =
    1
>> isa(y{1},'cell')      % contents of y is NOT a cell
ans =
    0
```

用户虽然可以同时显示两个或两个以上的数组单元的值，但不能将两个或两个以上数组单元的值赋给同一个变量。下边的例子试图将 B 的第二列（两个单元）的值全部赋给 d，Matlab 给出了错误提示：

```
>> B{:,2}
ans =
John Smith
ans =
    5
>> d = B{:,2}
??? Illegal right hand side in assignment. Too many elements.
```

请读者考虑一下，上面的代码为什么会出错？假如我们可以将上述 B 的两个单元的值赋给 d，那么 d 就是一个 2×1 的数组，其第一行元素是一个字符串，而第二行元素是一个标量，大家觉得这可能吗？有一个问题留给大家：如果将上例中的最后一行代码改为 d=B(:,2)，会得到什么样的结果呢？

当用户在单一的单元中取值时，可以将取值的下标范围附加在该单元索引后面，从而得到单元内部分内容值。下面的代码分别获取单元数组 A 的各单元的部分内容值（在取值之前，首先利用 celldisp 函数将 A 中所有单元的内容显示出来）：

```
>> A % recall prior data
A =
            [3×3 double]    [2.0000 + 3.0000i]
    'A character string'          [1×7 double]
>> celldisp(A) % display contents
A{1,1} =
    1    2    3
    4    5    6
    7    8    9
A{2,1} =
A character string
A{1,2} =
    2.0000 + 3.0000i
A{2,2} =
    12   10    8    6    4    2    0
>> A{1,1}(3,:)      % third row of 3-by-3 array
ans =
    7    8    9
>> A{4}(2:5)        % second through fifth elements of A{2,2}
ans =
    10    8    6    4
>> A{1,2}(2)        % second element doesn't exist
??? Index exceeds matrix dimensions.
>> A{2,1}(3:11)     % extract part of the character string
ans =
character
```

8.4　逗号分隔列表

逗号分隔列表（Comma-Separated Lists）是 Matlab 的一个重要语法概念。它通常将变量或常量用逗号隔开，形成一个列表，作为函数或命令的参数，目的是同时创建或获取两个或两个以上的数据单元（如前几章讲的维或本章讲到的单元和结构体）。

逗号分隔列表可以用于创建数组。下例中的代码，除了第三条语句以外，均使用了逗号分隔列表：

```
>> a = ones(2,3);
>> b = zeros(2,1);
>> c = (3:4)';
>> d = [a,b,c] % same as [a b c]
d =
    1    1    1    0    3
    1    1    1    0    4
```

另外，在函数的输入和输出参数中也可以使用逗号分隔列表，如下面的代码所示：

```
>> d = cat(2,a,b,c)
d =
    1    1    1    0    3
    1    1    1    0    4
>> [m,n] = size(d)
m =
    2
n =
    5
```

在单元数组中，我们只要将上面的逗号分隔列表中的变量或常量值用相应的数组单元的索引代替，就可以实现多个数组单元的创建或提取。我们来看下面的代码：

```
>> F = {a b c} % create a cell array
F =
    [2×3 double]    [2×1 double]    [2×1 double]
>> d = cat(2,F{:}) % same as cat(2,a,b,c)
d =
    1    1    1    0    3
    1    1    1    0    4
>> d = cat(2,F(:)) % not content addressing
d =
    [2×3 double]
    [2×1 double]
    [2×1 double]
```

在上面的代码中，cat(2,F{:})将被解释成 cat(2,F{1},F{2},F{3})，因此，cat 函数的输入参数实际上就是一个逗号分隔列表：2,F{1},F{2},F{3}。注意，只有在“按内容索引”时，程序才被解释成逗号分隔列表。在上例的最后一条语句中，单元索引是基于单元本身而不是它们的内容的，即是“按单元索引”的。在这种情况下，cat(2,F(:))不会被解释成逗号分隔列表。我们再考虑下面的代码：

```
>> d = [F{:}]
d =
    1    1    1    0    3
    1    1    1    0    4
>> d = [F{1}, F{2}, F{3}] % what is implied by the above
d =
```

```
     1     1     1     0     3
     1     1     1     0     4
>> e = [F{2:3}] % can also content address any subset
e =
     0     3
     0     4
>> e = [F{2},F{3}] % what is implied by the above
e =
     0     3
     0     4
```

刚开始时，用户可能会觉得逗号分隔列表比较难理解。但是，当对它比较熟悉之后，用户就会发现其功能的强大。有关更多逗号分隔列表的信息，请查看联机帮助文档中的 lists 或在联机帮助文档中查找 comma separated list。

利用逗号分隔列表语法，函数 deal 可以将多个单元的值提取出来赋给一个独立的单元数组变量。如果用户玩过“桥牌”，就知道“出牌”意味着将自己手中的牌按某种约定的方式排列出来。函数 deal 就相当于一个“出牌”的过程，它将单元数组的内容输出到一个个独立的变量中，如下面的代码所示：

```
>> celldisp(F) % recall data
F{1} =
     1     1     1
     1     1     1
F{2} =
     0
     0
F{3} =
     3
     4
>> [r,s,t] = deal(F{:}) % deal out contents of F
r =
     1     1     1
     1     1     1
s =
     0
     0
t =
     3
     4
```

上例中的变量 r、s 和 t 都是数值型变量，其中 r=F{1}，s=F{2}，t=F{3}。上边的示例代码其实等价于：

```
>> [r,s,t] = deal(F{1},F{2},F{3})
r =
     1     1     1
     1     1     1
s =
     0
     0
```

```
t =
     3
     4
```

上述 deal 函数的用法实际上很简单，它把第一个输入参数赋给第一个输出参数，将第二个输入参数赋给第二个输出参数，依次类推。虽然 deal 函数的用法很简单，但利用它可以用一条非常简单的语句从多个单元中提取数据并赋值给不同的变量。

由于 deal 函数的输出也可以是一个逗号分隔列表，因此，deal 函数也可以用来在一条语句中给多个单元赋值，如下面的代码所示：

```
>> [G{:}] = deal(a,b,c)
??? Error using ==> deal
The number of outputs should match the number of inputs.
>> [G{1:3}] = deal(a,b,c)
G =
    [2×3 double]    [2×1 double]    [2×1 double]
>> F ={a b c}
F =
    [2×3 double]    [2×1 double]    [2×1 double]
>> isequal(F,G) % True since F = G
ans =
     1
```

第一条语句产生错误的原因是 G{:}没有指定有多少单元数组元素要用逗号分隔。第二条语句指定了输出的单元数组元素为 3 个，因此，程序给出了正确结果。在第三条语句中，{a b c}已经生成了一个单元数组，所以就没有必要使用 deal 函数通过复制的方式创建单元数组了（而且用 deal 函数来完成这一操作速度要慢一些）。

8.5 单元数组函数

除了前几节讲到的 celldisp、cell、iscell、isa 和 deal 之外，Matlab 还提供了其他一些有用的函数用于处理单元数组。下面就一些重要的函数举例说明。

函数 cellfun 将一个指定的函数应用到一个单元数组的所有单元上，这样就不用针对每个单元调用函数了。下边的例子分别将 isreal、length、prodofsize、isclass 等函数应用到单元数组 A 上：

```
>> A % recall data
A =
            [3×3 double]        [2.0000+3.0000i]
    'A character string'            [1×7 double]
>> cellfun('isreal',A) % True=1 where not complex
ans =
     1     0
     1     1
>> cellfun('length',A) % length of contents
ans =
```

```
     3   1
    18   7
>> cellfun('prodofsize',A) % number of elements in each cell
ans =
     9   1
    18   7
>> cellfun('isclass',A,'char') % True for character strings
ans =
    0    0
    1    0
```

除了上述函数外，函数 cellfun 还可以将其他函数用于一个单元数组，对这些函数的应用请读者查阅相关帮助信息。

另一个在单元数组中应用比较广泛的函数是 num2cell 函数。该函数从一个数组（可以是任何数据类型）中提取指定的元素，然后填充到一个单元数组中。下例创建了一个随机数组 a，然后利用 num2cell 函数提取 a 的相应元素构造单元数组 c、d 和 e：

```
>> a = rand(3,6)    % new numerical data
a =
    0.7266      0.2679      0.6833      0.6288      0.6072      0.5751
    0.4120      0.4399      0.2126      0.1338      0.6299      0.4514
    0.7446      0.9334      0.8392      0.2071      0.3705      0.0439
>> c = num2cell(a)  % c{i,j}=a(i,j)
c =
    [0.7266]    [0.2679]    [0.6833]    [0.6288]    [0.6072]    [0.5751]
    [0.4120]    [0.4399]    [0.2126]    [0.1338]    [0.6299]    [0.4514]
    [0.7446]    [0.9334]    [0.8392]    [0.2071]    [0.3705]    [0.0439]
>> d = num2cell(a,1) % d{i}= a(:,i)
d =
    Columns 1 through 4
     [3×1 double]   [3×1 double]    [3×1 double]    [3×1 double]
    Columms 5 through 6
     [3×1 double]   [3×1 double]
>> e = num2cell(a,2) % e{i}=a(i,:)
e =
    [1×6 double]
    [1×6 double]
    [1×6 double]
```

需要指出的是，虽然上例中的 a 是数值数组，但 num2cell 函数支持各种类型的数组，例如字符串数组、逻辑数组等。另外，在 num2cell 函数的上述 3 种应用情况中，后两种应用的使用频率要远高于第一种。

8.6　字符串单元数组

根据本书安排，有关字符串的详细讨论将在下一章给出，本章我们主要讨论字符串的用法及其在单元数组中的存储问题。读者也可以先阅读下一章，再回过头来看本章下边的内容。

在 Matlab 中，字符串是一段由单引号括起来的字符序列。下例生成了一个字符串并将其存储为变量 s：

```
>> s = 'Rarely is the question asked: is our children learning?'
s =
Rarely is the question asked: is our children learning?
```

在很多应用场合，某些字符串之间是有相互关联的。这时，我们可以将这些字符串储存在一个单元数组中，这样要比将这些字符串保存成一个个单独的变量使用起来方便。下例将 s 与另外两个字符串同时存储到了单元数组 cs 中：

```
>> cs ={'My answer is bring them on.'
s
'I understand small business growth. I was one.'}
cs =
    'My answer is bring them on.'
    'Rarely is the question asked: is our children learning?'
    'I understand small business growth. I was one.'
>> size(cs)     % a column cell array
ans =
    3   1
>> iscell(cs)   % yes, it is a cell array
ans =
    1
```

上例中，单元数组 cs 有 3 个单元，每个单元都由一个字符串构成。在 Matlab 中，我们称这种类型的单元数组为字符串单元数组。字符串单元数组是 Matlab 中应用十分广泛的一个重要数据类型，Matlab 专门提供了几个函数用于处理这类数组。下面的代码判断一个单元数组 cs 是否为字符串单元数组：

```
>> iscellstr(cs)
ans =
    1
```

对于 iscellstr 函数，只有当输入的单元数组的所有单元均由字符串构成时，函数才返回 True=1，否则就返回 False=0。

在 Matlab 引入单元数组之前，相关的字符串通常存储在一个字符串数组中，其中每个字符串在该数组中占据单独的一行，有几个字符串，数组就有几行，数组的列数由其中最长的那个字符串决定，没有字符的位置将被填充为空白字符。字符串数组中的每一个字符都占据一个单独的位置，并且可以像一个数值数组那样进行索引。

Matlab 提供了两个函数 char 和 cellstr，实现一个字符串单元数组和一个字符串数组之间的转换。其中 char 函数将一个字符串单元数组转换成一个字符串数组；cellstr 函数将一个字符串数组转换成一个字符串单元数组。下例则利用 char 将字符串单元数组 cs 转换为字符串数组 sa，又利用 cellstr 将 sa 转换为字符串单元数组 cst，并证明 cs 与 cst 是相同的：

```
>> cs % recall previous cell array of strings
cs =
    'My answer is bring them on.'
```

```
    'Rarely is the question asked: is our children learning?'
    'I understand small business growth. I was one.'
>> sa = char(cs) % convert to a string array
sa =
    My answer is bring them on.
    Rarely is the question asked: is our children learning?
    I understand small business growth. I was one.
>> ischar(sa)       % True for string array
ans =
    1
>> iscell(sa)       % True for cell array
ans =
    0
>> size(sa)         % size of string array
ans =
    3   34
>> size(cs)         % size of cell array
ans =
    3   1
>> cst = cellstr(sa) % convert back to cell array
cst =
    'My answer is bring them on.'
    'Rarely is the question asked: is our children learning?'
    'I understand small business growth. I was one.'
>> iscell(cst)      % True for cell array
ans =
    1
>> isequal(cs,cst)  % True for equal variables
ans =
    1
>> isequal(cs,sa)   % cell array not equal to string array
ans =
    0
```

有上述代码结果可以看出，char 函数和 cellstr 函数执行的是互逆操作，执行结果之间可以相互转换。另外，由于一个字符串数组的每一行都必须有相同的列数，因此，Matlab 会根据需要在没有字符填充的位置添加空白字符，以便使字符串数组成为一个矩形数组。

8.7 结构体的创建

结构体（structures）在很大程度上与单元数组非常相似，它也允许用户将类型不同的数据集中在一个单独的变量中。与单元数组不同的是，结构体是用称之为字段（fields）的名称来对其元素进行索引的，而不是通过数字索引。另外，从原理上讲，Matlab 也可以创建任意维数的结构体，但在大多数情况下，为了处理方便，用户通常只需要创建一个简单的结构体向量（即一维的结构体）即可。

结构体与单元数组不同，它采用点号来访问字段中的数据变量，这一点与 C++语言中的类有些相似。我们只要采用点号，为结构体中的各个字段赋上初值，就创建了这个结构体。例如，下例创建了结构体变量 circle：

```
>> circle.radius =2.5;      % semicolon, no display
>> circle.center = [0,1];
>> circle.linestyle = '--';
>> circle.color = 'red'     % no semicolon, so display
circle =
    radius: 2.5
    center: [0 1]
    linestyle: '--'
    color: 'red'
```

由上边的代码可知，结构体变量 circle 包括 radius、center、linestyle 和 color 4 个字段，分别代表一个圆的半径、圆心、线型和颜色。请读者注意，字段名是区分大小写的，即 radius 与 Radius 是不同的字段，另外，结构体字段名的命名规则与普通变量名的命名规则是相同的，例如，一个字段名最多只能包含 31 个字符，字段名必须用字母开头等。

利用 size 函数和 whos 函数都可以查看结构体的大小，不过 whos 函数除了显示维数大小外，还显示变量的字节数和变量类型，如下例所示：

```
>> size(circle)
ans =
    1   1
>> whos
    Name        Size        Bytes       Class
    ans         1×2         16          double array
    circle      1×1         530         struct array
Grand total is 14 elements using 546 bytes
```

上面的结果表明，circle 是一个结构体标量（因为其维数是 1×1），也就是说，该结构体内只包含了一个圆的信息。如果用户想在该结构体再创建一个圆，那么可以将这个圆存储为 circle 变量的第二个元素，其代码如下所示：

```
>> circle(2).radius = 3.4;
>> circle(2).color = 'green';
>> circle(2).linestyle = ':';
>> circle(2).center = [2.3 -1.2]
circle =
1×2 struct array with fields:
    radius
    center
    linestyle
    color
```

第二个圆创建完成后，circle 就变成了含有两个元素的结构体变量。我们可以通过上例总结在结构体中添加新元素的一般方法：首先必须在变量名后边标明新元素的索引值（如本例中的 2，表明添加第二个元素）；然后利用“.字段名”语法为一个或多个字段赋值（如本例中利用.radius 为第二个圆的半径赋值）。在这里有两点请读者注意：①在为结构体中的各个字段赋值时，顺序可以不同，也就是说，先为半径赋值还是先为颜色赋值没有顺序要求。②即使对同一个字段，每次赋值时可以采用不同的数据类型，例如，我们既可以将一个数值赋给 radius 字段，也可以将一个字符串赋给它。例如，下面的代码将'sqrt(2)'这一字

符串赋给第二个圆的半径，同样是合法的：

```
>> circle(2).radius = 'sqrt(2)'
circle =
1×2 struct array with fields:
    radius
    center
    linestyle
    color
>> circle.radius % display radius contents
ans =
    2.5
ans =
sqrt(2)
```

从显示结果可以看出，circle(1).radius 是一个数值类型的数据，而 circle(2).radius 是一个字符串。

也许仅从上面的论述，读者还是不明白结构体到底有什么用。我们不妨反过来考虑，假如不用结构体，这些数据该怎么存储。一个通用的办法是为每一个字段设置一个数组变量，分别存储相应的数据，例如，我们可以设置 4 个数组变量分别存储各个圆的半径、圆心、线型和颜色，如下例所示：

```
>> Cradius = [2.5 3.4]; % ignore sqrt(2) change above
>> Ccenter = [0 1 ; 2.3 -1.2];
>> Clinestyle = {'--' ':'}; % cell array of strings
>> Ccolor = {'red' 'green'};
```

当采用上面的数据格式时，用户要想获得一个圆的信息，必须从这 4 个变量中分别取值，这或许还不是最麻烦的事情，如果用户想要向这些变量中添加另一个圆的信息，就需要分别对这些变量进行赋值，这显然是一件非常繁琐的事情。下面的例子分别采用结构体方法和数组变量方法添加第三个圆的信息，读者很容易看出哪种方法更加快捷方便：

```
>> circle(3).radius = 25.4;
>> circle(3).center = [-1 0];
>> circle(3).linestyle = '-.';
>> circle(3).color = 'blue'    % third circle added
circle =
1×3 struct array with fields:
    radius
    center
    linstyle
    color
>> Cradius(3) = 25.4
Cradius =
    2.5     3.4     25.4
>> Ccenter(3,:) = [-1 0]
Ccenter =
      0         1
    2.3      -1.2
```

```
    -1         0
>> Cradius(3) = 25.4
Cradius =
    2.5     3.4     25.4
>> Clinestyle{3} = '-.'
Clinestyle =
    '--'    ':'     '-.'
>> Ccolor(3) = {'blue'}
Ccolor =
    'red'   'green' 'blue'
```

另外，结构体还有一个方便之处，就是函数的参数传递问题。例如，如果圆的信息存储在一个结构体中，那么一个函数要想获得该圆的信息，只需将该圆所在的结构体作为参数传递给函数就可以了，即 myfunc(circle)；如果采用前面讲的数组变量方法存储这个圆的信息，那么一个函数要想获得该圆的信息，就需要传递 4 个参数，即 myfunc (Cradius, Ccenter, Clinestyle, Ccolor)。

利用结构体，用户可以很方便地添加一个新字段。下面的代码向 circle 中添加新字段 filled：

```
>> circle(1).filled = 'yes'
circle =
1×3 struct array with fields;
    radius
    center
    linestyle
    color
    filled
>> circle.filled % display all .filled fields
ans =
yes
ans =
    []
ans =
    []
```

当结构体向一个元素添加新字段时，其他元素也自动获得该字段属性，例如，上例虽然只向第一个圆添加了 filled（是否填充）字段，但其他两个圆也获得了 filled 字段。由于后两个 filled 没有被赋值，因此被设为空矩阵[ ]，给这些字段赋值也是非常容易的事，如下例所示：

```
>> circle(2).filled = 'no';
>> circle(3).filled = 'yes';
>> circle.filled
ans =
yes
ans =
no
ans =
yes
```

当用户不喜欢用直接赋值的方式创建结构体的时候，也可以使用 Matlab 提供的 struct 函数创建一个结构体。下面的代码利用 struct 函数创建了一个与 circle 相同的结构体 CIRCLE：

```
>> values1 = {2.5 'sqrt(2)' 25.4}; % cell arrays with field data
>> values2 = {[0 1] [2.3 -1.2] [-1 0]};
>> values3 = {'--' ':' '-.'};
>> values4 = {'red' 'green' 'blue'};
>> values5 = {'yes' 'no' 'yes'};
>> CIRCLE = struct('radius',values1,'center',values2,…
           'linestyle',values3,'color',values4,'filled',values5)
CIRCLE =
1×3 struct array with fields:
        radius
        center
        linestyle
        color
        filled
>> isequal(circle,CIRCLE) % True since structures are equal
ans =
    1
```

8.8 结构体的处理

结构体作为一种特殊的数组类型，具有与数值型数组和单元数组相同的处理方式，例如同样可以进行组合和索引。在组合结构体时，进行组合的结构体必须具有相同的字段。例如，下面的代码将上节创建的 circle 和 CIRCLE 组合成 A，但新创建的 square 却不能和 circle 组合：

```
>> square.width = 5; % a new structure
>> square.height = 14;
>> square.center = zeros(1,2);
>> square.rotation = pi/4
square =
       width: 5
      height: 14
      center: [0 0]
    rotation: 0.7854
>> A = [circle CIRCLE]
A =
1×6 struct array with fields:
    radius
    center
    linestyle
    color
    filled
>> B = [circle square]
??? Number of fields does not match in [] concatenation.
```

因为 circle 和 CIRCLE 都是 1×3 的结构体，并且具有相同的字段，因此将二者组合生成的结构体 A 便是一个 1×6 的结构体。另外，由于结构体 square 和 circle 的字段不完全相同，因此，无法将二者进行组合生成 B。

用户可以通过直接索引获取一个结构体数组的子数组。例如，下例分别获取 circle 和 CIRCLE 的子数组并组合成 C：

```
>> C = [circle(1:2) CIRCLE(3)]
C =
1×3 struct array with fields:
    radius
    center
    linestyle
    color
    filled
>> isequal(C,circle) % True since equal
ans =
    1
```

由于 CIRCLE 和 circle 是两个相同的结构体，因此，从各字段内容上看，C 和 circle 是完全相同的。

结构转换函数 reshape 和复制函数 repmat 同样可以对结构体进行处理，但我们通常情况下不怎么使用这两个函数。下面的代码分别将 A（1×6）转换为 3×2 和 1×2×3 的结构体：

```
>> Aa = reshape(A,3,2)
Aa =
3×2 struct array with fields:
    radius
    center
    linestyle
    color
    filled
>> Aaa = reshape(A,1,2,3)
Aaa =
1×2×3 struct array with fields:
    radius
    center
    linestyle
    color
    filled
```

在实际应用中，结构体多是用向量格式表示，很少出现多维的情况，因此 reshape 函数在结构体中很少使用。

repmat 函数可用来从一个结构体生成另一个结构体，原结构体中各字段的内容将复制到新的结构体中。下例从 square 结构体（1×1）生成一个 3×1 的结构体 S，并且 S 中的每个元素都与 square 相同：

```
>> S = repmat(square,3,1)
```

```
S =
3×1 struct array with fields:
    width
    height
    center
    rotation
>> S.width % look at all width fields
ans =
    5
ans =
    5
ans =
    5
```

当用 repmat 函数创建结构体时，Matlab 自动将原结构体内各字段的值复制到创建的结构体的各个元素中。因此，上例中 S 包含 3 个相同的元素，并且都与 square 具有相同的字段值。结构体被创建后，用户可以根据需要修正各元素的值，以便进行不同的操作。

8.9 结构体内容的获取

当用户知道一个结构体中某个字段的名称时，要获取指定结构体元素中该字段的数据，只需利用点号标识出该字段的名字就可以了。例如，下面的代码获取 circle 中第一和第二个元素的 radius 值：

```
>> rad2 = circle(2).radius
rad2 =
sqrt(2)

>> circle(1).radius
ans =
      2.5

>> areal = pi*circle(1).radius^2
areal =
    19.635
```

第三条语句获取 circle 中第一个元素的 radius 值，并用它来计算该圆的面积。

当结构体中某个字段的值是一个数组时，在字段名后添加直接数组索引，就可以获得该字段的一个子数组。例如，circle 中的 filled 字段是字符串值，下面的代码获取 circle(1) 中该字段的全部和部分值：

```
>> circle(1).filled         % the entire field
ans =
yes
>> circle(1).filled(1)      % first element of field
ans =
y
>> circle(1).filled(2:end)  % rest of field
ans =
es
```

和单元数组一样，用户也不能将多个结构体元素或字段的值赋给一个变量。例如，下面的赋值是非法的：

```
>> col = circle.color
??? Illegal right hand side in assignment. Too many elements.
```

上边这条语句试图从结构 circle 中获取 3 个字段的数据并保存在一个变量中，Matlab 给出了出错信息。要解决这一问题，用户可以采用后面讲到的逗号分隔列表和 deal 函数。

由于结构体的字段名是用与变量类似的命名规则命名的，因此，我们可以先将字段名存储为一个字符串变量，然后通过引用该变量获得字段中的内容，在 Matlab 中，这种方法通常称为动态寻址。例如，下面的代码采用动态寻址获取 circle 中 color 和 radius 字段的内容：

```
>> fldstr = 'color';    % store desired field in variable

>> circle.(fldstr)      % get all color fields
ans =
red
ans =
green
ans =
blue

>> fldstr = 'radius';
area = pi*circle(1).(fldstr)^2    % compute area of first circle
area =
       19.635
```

动态寻址与普通的下标寻址类似，二者均需用到圆括号，只不过前者用一个字符串变量代表地址，后者用数字代表地址。

8.10 逗号分隔列表

前文介绍的逗号分隔列表语法同样可以用于同时获得多个结构体数组元素的值。其用法与单元数组的情况相同，也是将变量或常量用逗号隔开，形成一个列表，作为函数或命令的参数使用。逗号分隔列表可以用于数组创建，如下面的代码所示：

```
>> a = ones(2,3);
>> b = zeros(2,1);
>> c = (3:4)';
>> d = [a,b,c] % same as [a b c]
d =
     1     1     1     0     3
     1     1     1     0     4
```

逗号分隔列表还出现在函数的输入和输出参数列表中，如下面的代码所示：

```
>> d = cat(2,a,b,c)
d =
     1     1     1     0     3
```

```
    1    1    1    0    4
>> [m,n] = size(d)
m =
    2
n =
    5
```

根据上面的阐述，在结构体中使用逗号分隔列表的用法如下：将一个附带字段名的结构体数组放在逗号分隔列表的位置，Matlab 将会把字段的内容取出，并将其用逗号隔开，依次排列。我们来看下面的代码：

```
>> cent = cat(1,circle.center) % comma separated list syntax
cent =
             0            1
          2.3         -1.2
            -1            0
>> cent = cat(1,circle(1).center,circle(2).center,circle(3).center)
cent =
             0            1
          2.3         -1.2
            -1            0
>> some = cat(1,circle(2:end).center)
some =
          2.3         -1.2
           -1            0
```

上述语句中，cat 函数用于将圆心的值作为一行附加到数值型数组 cent 中。上例中，前两条语句的逗号分隔列表语法是等效的。第三条语句表明用户还可以利用逗号分隔列表索引一个子数组，并提取这个子数组的值。

因为结构体 circle 的 color 字段包含了不同长度的字符串，因此这些字段不能赋值给一个字符串数组，但可以赋值给一个单元数组，如下面的代码所示：

```
>> circle.color
ans =
red
ans =
green
ans =
blue

>> col = cat(1,circle.color) % elements have different lengths
??? Error using ==> cat
CAT arguments dimensions are not consistent.
>> col = [circle.color] % no error but not much use!
col =
redgreenblue
>> col = {circle.color} % cell array of strings
col =
    'red'       'green'      'blue'
>> col = char(col) % if needed, convert to string array
```

```
col =
red
green
blue
```

Matlab 没有提供提取结构体数组元素所有字段的函数。例如，用户不能用一条命令来同时获取第一个圆的 radius、center、linestyle、color 和 filled 字段。从某种意义上来说，这些函数的存在是没有必要的，因为所有的字段都可以通过直接寻址获得并用于计算，例如，areal=pi*circle(1).radius^2。

利用逗号分隔列表语法，函数 deal 可以将多个结构体元素的值提取出来并赋值给不同的变量，如下面的代码所示：

```
>> [c1,c2,c3] = deal(circle.color) % get all colors
c1 =
red
c2 =
green
c3=
blue
>> [rad1,rad3]=deal(circle([1 3]).radius) % 1st and 3rd radius
rad1 =
        2.5
rad3 =
       25.4
```

由于 deal 函数的输出也可以是一个逗号分隔列表，因此，deal 函数也可以用来在一条语句中给多个结构体数组元素的同一个字段赋值，如下面的代码所示：

```
>> [circle.radius] = deal(5,14,83)
circle =
1×3 struct array with fields:
    radius
    center
    linestyle
    color
    filled
>> circle.radius % confirm assignments
ans =
     5
ans =
    14
ans =
    83
>> [triangle(:).type] = deal('right','isosceles','unknown')
??? Error using ==> deal
The number of outputs should match the number of inputs.
>> [triangle(1:3).type] = deal('right','isosceles','unknown')
triangle =
1×3 struct array with fields:
    type
```

```
>> triangle.type
ans =
right
ans =
isosceles
ans =
unknown
```

在上述第一条语句中，由于结构体 circle 已经存在并且有 3 个元素，因此，该语句的输出参数将被扩展成 3 个元素。在第二条语句中，由于结构体 triangle 并不存在，因此，该语句返回一条出错信息，因为它无法确定在一个逗号分隔列表中需要生成多少个元素。在最后一条语句中，由于指定了元素的个数，因此，程序就使用给定的数据填充了新创建的结构体 triangle 的 type 字段。

8.11　结构体函数

当一个函数调用结构体时，它首先要知道该结构体各字段的名称。在 Matlab 命令窗口中，显示结构体各字段的名称很简单，只需直接输入结构体的名称即可。例如，下面的代码显示结构体 circle 和 square 的各字段名称：

```
>> circle
circle =
1×3 struct array with fields:
    radius
    center
    linestyle
    color
    filled
>> square
square =
    width:5
    height:14
    center:[0 0]
    rotation:0.7854
```

注意：由于 square 只有一个元素，因此各字段的内容也被显示了出来。

当我们把一个结构体传递给一个函数的时候，例如，myfunc(circle)，该函数必须知道如何访问结构体中的字段，否则就无法从这个结构体中获取数据。在用户编写 myfunc 函数时，通常会用到函数 fieldnames，该函数返回结构体的字段名，例如，下例返回 circle 的各字段的名字：

```
>> fieldnames(circle)
ans =
    'radius'
    'center'
    'linestyle'
    'color'
    'filled'
```

由上面的结果可以看出，fieldnames 函数的输出是一个字符串单元数组，该数组由结构体的各字段的字段名组成。

另一个常用的函数是 isfield 函数，它用来判断一个字段名是否为指定结构体中的字段名。例如，下例判断出 color 是 circle 的字段名，而 width 不是：

```
>> isfield(circle,'color') % True
ans =
    1
>> isfield(circle,'width') % False
ans =
    0
```

如果用户想知道一个变量是否是结构体变量，可以用函数 class 和 isstruct 来判断。例如，下面的代码判断出 square 和 circle 是结构体，而标量 d 不是结构体：

```
>> class(square)          % ask for the class of variable square
ans =
struct
>> isstruct(circle)       % True for structures
ans =
    1
>> d = pi;
>> isstruct(d)            % False for doubles
ans =
    0
```

如果用户不需要一个结构体中的某个字段，可以使用 rmfield 删除它。下面的代码给出了该函数的具体应用。

在使用 rmfield 函数之前，我们需要将 circle 的字段名存储在一个单元数组中，代码如下：

```
>> fnames = fieldnames(circle)
fnames =
    'radius'
    'center'
    'linestyle'
    'color'
    'filled'
```

如果我们想将 filled 字段从结构体 circle 中删除，并将删除后的结果赋给结构体 circle2，可以使用如下代码：

```
>> circle2 = rmfield(circle,fnames{5})
circle2 =
1×3 struct array with fields:
    radius
    center
    linestyle
    color
```

如果我们想同时删除多个字段，可以使用冒号索引。例如，下面的代码删除 circle 的前三个字段，并将删除后的结果赋给结构体 circle3：

```
>> circle3 = rmfield(circle,{fnames{1:3}})
circle3 =
1×3 struct array with fields:
    'color'
    'filled'
```

有时候，我们需要重新排列结构体中各字段的顺序，orderfields 函数可以完成这一功能。例如，下面的代码将 circle 中的各字段按照 ASCII 码的顺序重新排列，并将结果赋给 circleA：

```
>> circleA = orderfields(circle)
circleA =
1×3 struct array with fields:
    center
    color
    filled
    linestyle
    radius
```

除了按 ASCII 码顺序排列外，还可以按照用户指定的顺序将结构体中的字段进行排序。例如，下面的代码将 circleA 中的各字段按照与 CIRCLE 相同的顺序排列，并将结果赋给 circleB：

```
>> circleB = orderfields(circleA,CIRCLE)  % match fields of structure CIRCLE
circleB =
1×3 struct array with fields:
    radius
    center
    linestyle
    color
    filled
```

下面的代码则将 circle 中的各字段按照用户指定的顺序排列，并将结果赋给 circleC：

```
>> circleC = orderfields(circle,[2 5 1 4 3])  % provide permutation vector
circleC =
1×3 struct array with fields:
    center
    filled
    radius
    color
    linestyle
```

下面的代码则将 circle 中的各字段按逆序排列，并将结果赋给 circleD：

```
>> circleD= orderfields(circle,fnames(end:-1:1))  % reverse original
order
circleD =
1×3 struct array with fields:
    filled
```

```
color
linestyle
center
radius
```

最后，鉴于结构体和单元数组在数值运算中的重要作用，加之二者具有很多相似性，Matlab 专门提供了两个函数 cell2struct 和 struct2cell 来实现它们之间的转换。这两个函数的功能和使用方法都比较简单，本书不再赘述，有兴趣的读者可以参考相关的帮助文档。

8.12 小结

本章讲到的单元数组和结构体是两种功能十分强大的承载型数据类型，它们可以存储任何类型的数据，甚至包括它们本身。单元数组和结构体之间可以实现任意深度的相互嵌套。例如，我们可以创建这样一个单元数组变量：该数组的一个单元中包含一个结构体，而这个结构体中的一个字段中又包含另一个结构体，这个结构体又包含一个字段，该字段包含一个单元数组，这个单元数组的单元中又包含另一个单元数组。或许读者此时已被这种错综复杂的嵌套关系搞糊涂了，但大家必须相信，在 Matlab 中创建这样一个变量的确是可能的。因此，无需我们多言，相信用户已经看到了单元数组和结构体无比强大的功能以及在实际应用中的广泛应用。作为本章的结束，请大家根据下面的代码思考后面的问题：

```
>> one(2).three(4).five = {circle}
one =
1×2 struct array with fields:
    three
>> test = {{{circle}}}
test =
{1×1 cell}
```

第一条语句中究竟涉及到了多少个结构体？该语句总共生成多少结构体元素（包括空数组）？用一条命令能够从第二条语句中提取出结构体 circle 吗？

Chapter 9

字　符　串

尽管 Matlab 的主要功能在于对数值的处理能力，但有时我们也不可避免地遇到处理文本的情形，例如在画图时需要插入坐标轴标签和标题等。鉴于此，Matlab 也定义了针对文本处理的数据类型——字符串。

9.1 字符串结构

Matlab 中，字符串是特殊的 ASCII 数值型数组，只不过它们显示出来的是字符形式。下例创建了一个字符串 t，并用 size 和 whos 函数查看其大小：

```
>> t = 'How about this character string?'
t =
How about this character string?
>> size(t)
ans =
    1   32
>> whos
Name          Size          Bytes         Class
ans           1×2           16            double array
t             1×32          64            char array

Grand total is 34 elements using 80 bytes
```

在 Matlab 中，一个字符串就是用单引号括起来的一系列字符的组合，其中的每个字符都是该字符串的一个元素，通常都用两个字节来存储。

要查看一个字符串的底层 ASCII 值，用户只要使用一个简单的数学运算函数（例如，double、abs 等）就可以了。下例利用 double 和 abs 函数查看了 t 的 ASCII 值：

```
>> u = double(t)
u =
    Columns 1 through 12
      72    111   119    32   97    98   111    117    116    32       116    104
    Columns 13 through 24
      105    115    32     99  104    97   114    97     99     116      101    114
    Columns 25 through 32
```

```
     32    115   116    114 105   110  103   63
>> abs(t)
ans =
    Columns 1 through 12
      72    111     119   32    97    98  111    117   116   32   116   104
    Columns 13 through 24
      105   115     32    99    104   97  114    97    99   116   101   114
    Columns 25 through 32
      32    115     116   114   105  110   103   63
```

要想将一串 ASCII 值转化为字符串显示，可以使用 char 函数，下例将 u 重新转化为字符串进行显示：

```
>> char(u)
ans =
How about this character string?
```

注意：Matlab 可以把一个负值转换成一个空字符，但同时给出一条警告信息；对于大于 255 的数，Matlab 将直接从大于 char(255)的字符中选取（这一点与以前的版本不同）。读者可以利用下面的代码验证这两条结论：

```
>> a = double('a')
a =
    97

>> char(a)
ans =
a

>> char(a+256) % adding 256 does change the result
ans =
š

>> char(a-256) % negative value produces a blank character
Warning: Out of range or non-integer values truncated during conversion
to character.
ans =
```

因为字符串从其本质讲是用 ASCII 值表示的数组，因此它们可以用 Matlab 提供的所有数组处理工具进行处理。例如，下面的代码利用直接数组索引获取字符串中的部分字符：

```
>> u = t(16:24)
u =
character
```

上例中，由于 t 的第 16 到 24 个元素包含的是 character 这个词，因此，u 的值为'character'。下例则按照相反的顺序获取 t 的第 16 到 24 个元素：

```
>> u = t(24:-1:16)
u =
retcarahc
```

我们还可以使用转置操作符将字符串转化为列的形式。如，下例获取 t 的第 16 到 24 个元素并写成列的形式：

```
>> u = t(16:24)'
u =
c
h
a
r
a
c
t
e
r
```

若一个字符串本身含有单引号这个字符，需要将该单引号写成两个单引号形式，如下例所示：

```
>> v = 'I can''t find the manual!'
v =
I can't find the manual!
```

另外，我们可以使用与数组组合相同的方式将多个字符串组合成一个字符串，如下例所示：

```
>> u = 'If a woodchuck could chuck wood,';
>> v =' how much wood could a woodchuck chuck?';

>> w = [u v]
w =
If a woodchuck could chuck wood, how much wood could a woodchuck chuck?
```

函数 disp 显示一个字符串的内容而不显示其变量名，因此，它通常用于在脚本文件中显示必要的说明性文字。例如，下例用 disp 函数显示了 u 的内容：

```
>> disp(u)
If a woodchuck could chuck wood,
```

字符串数组可以是多行多列的数组，但必须保证每一行都有相同的列数。因此，当用户创建一个多行字符串数组时，需要显式地输入部分空格，以便使所有的行长度相同。例如，下面的代码创建一个 3 行的字符串数组 v，其中的第二行和第三行都输入了部分空格以便使各行长度相等：

```
>> v = ['Character strings having more than'
        'one row must have the same number '
        'of columns just like arrays!      ']
v =
Character strings having more than
one row must have the same number
of columns just like arrays!
```

为了省去输入空格的麻烦，我们可以采用函数 char 和 strvcat 从多个长度不同的字符串生成一个多行字符串数组。例如，下面的代码分别利用函数 char 和 strvcat 由 6 个长度不同的单词生成一个 6 行的字符串数组 legends：

```
>> legends = char('Wilt', 'Russel', 'Kareem', 'Bird', 'Magic', 'Jordan')
legends =
Wilt
Russel
Kareem
Bird
Magic
Jordan
>> legends = strvcat('Wilt','Russel','Kareem','Bird','Magic','Jordan')
legends =
Wilt
Russel
Kareem
Bird
Magic
Jordan
>> size(legends)
ans =
    6   6
```

由上述代码可知，函数 char 和 strvcat 功能基本相同，惟一的区别在于 strvcat 函数忽略空字符串输入，而 char 函数遇到空字符串就插入一个空行，如下例所示：

```
>> char('one',' ','two','three')
ans =
one

two
three
>> strvcat('one',' ','two','three')
ans =
one
two
three
```

如果两个或多个字符串数组具有相同的行数，则可以使用 strcat 函数将它们在水平方向上连接起来。该函数在连接时首先将各行填补的空格符去掉，然后对剩余的有效字符进行连接，连接完成之后，再根据生成的字符串数组各行的需要，补充空格符。下边的例子使用 strcat 函数将字符串数组 a 和 b 水平连接起来：

```
>> a = char('apples','bananas')
a =
apples
bananas
>> b = char('oranges','grapefruit')
b =
oranges
grapefruit
>> strcat(a,b)
ans =
```

```
applesoranges
bananasgrapefruit
```

因为字符串数组为了保证各行相等，将较短的行用空格符补充，因此，当我们从一个字符串数组中提取一行时，有时会含有不需要的空格符，要想将这些空格符去掉，可以使用 **deblank** 函数。该函数的具体用法如下面的代码所示：

```
>> c = legends(4,:)
c =
Bird

>> size(c)
ans =
    1    6

>> c = deblank(legends(4,:))
c =
Bird

>> size(c)
ans =
    1    4
```

9.2　数字与字符串的相互转换

在很多情况下，我们需要把一个数值数组转换成一个字符串，或把一个字符串转换成一个数值数组。Matlab 提供了一系列函数完成这些操作，包括 int2str、num2str、mat2str、sprintf 和 fprintf。下边的代码给出了前 3 个函数的应用实例：

```
>> int2str(eye(3)) % convert integer arrays
ans =
1    0    0
0    1    0
0    0    1

>> size(ans) % it's a character array, not a numerical matrix
ans =
    3    7

>> num2str(rand(2,4)) % convert noninteger arrays
ans =
0.95013      0.60684      0.8913      0.45647
0.23114      0.48598      0.7621      0.01850

>> size(ans) % again it is a character array
ans =
    2       40

>> mat2str(pi*eye(2)) % convert to MATLAB input syntax form!
ans =
[3.14159265358979 0; 0 3.14159265358979]

>> size(ans)
```

```
ans =
    1        40
```

下面的代码给出了函数 fprintf 和 sprintf 的应用实例：

```
>> fprintf('%.4g\n',sqrt(2))    % display in Command window
1.414
>> sprintf('%.4g',sqrt(2))      % create character string
ans =
1.414

>> size(ans)
ans =
    1    5
```

函数 fprintf 和 sprintf 是通用格式转换函数，其名称和用法均类似于 C 语言中的格式转换函数（例如，printf）。这两个函数使得用户在处理和显示数据时具有了更大的灵活性。通常，fprintf 函数用于将数值型结果转换成 ASCII 字符格式，并将转换的结果附加在一个数据文件之后。但是，用户必须将一个文件标识符作为第一个参数传递给 fprintf 函数，否则该函数将把结果直接显示在命令窗口中。另外，如果用户将 1 作为文件标识符传递给该函数，它也会把结果直接显示在命令窗口中。sprintf 函数与 fprintf 基本相同，只不过它仅仅生成一个字符数组，该字符数组可以用于显示，也可以传递给另一个函数，还可以对其进行修改。由于 sprintf 函数和 fprintf 函数的用法几乎完全一致，因此，下面仅以 sprintf 为例阐述这两个函数的用法。首先我们看一个简单的例子：

```
>> radius = sqrt(2);
>> area = pi * radius^2;

>> s = sprintf('A circle of radius %.5g has an area of %.5g.',radius,area)
s =
A circle of radius 1.4142 has an area of 6.2832.
```

其中的%.5g 是变量 radius 在作为字符显示时的格式声明，它表明用户需要显示 radius 的 5 个有效位。

sprintf 经常用于为一个图表生成用作注释的字符串，在一个图形用户界面中显示数值型数据，或者创建函数数字编号的一系列数据文件等。下面的代码给出了用 sprintf 函数创建数据文件的例子：

```
>> i = 3;
>> fname = sprintf('mydata%.0f.dat',i)
fname =
mydata3.data
```

在 Matlab 5 之前的版本中，函数 int2str 和 num2str 仅仅使用%.0f 和%.4g 两个格式声明调用 sprintf 函数，从而将一个整数和实数转换为字符串。在 Matlab 5 中，int2str 和 num2str 的功能得到了增强，用户可以将它们用于数值型数组。例如，前面利用 sprintf 显示圆面积的例子也可以用下述方法实现：

```
s = ['A circle of radius ' num2str(radius) ' has an area of ' ...
num2str(area) '.']
```

```
s =
A circle of radius 1.4142 has an area of 6.2832.
```

虽然这条语句和前面所得到的结果相同，它却需要更多的计算量，并且容易出现拼写错误（例如漏写了空格或者单引号），其程序的可读性也比较差。因此，建议用户在此情况下尽量使用 sprintf 函数，避免使用 int2str 和 num2str 函数。

关于 sprintf 函数的其他用法和更详细的信息，请读者参考下面给出的该函数的帮助文档：

```
>> help sprintf
SPRINTF Write formatted data to string.
   [S,ERRMSG] = SPRINTF(FORMAT,A,...) formats the data in the real
   part of array A (and in any additional array arguments), under
   control of the specified FORMAT string, and returns it in the
   MATLAB string variable S. ERRMSG is an optional output argument
   that returns an error message string if an error occurred or an
   empty string if an error did not occur. SPRINTF is the same as
   FPRINTF except that it returns the data in a MATLAB string
   variable rather than writing it to a file.

   FORMAT is a string containing C language conversion specifications.
   Conversion specifications involve the character %, optional flags,
   optional width and precision fields, optional subtype specifier, and
   conversion characters d, i, o, u, x, X, f, e, E, g, G, c, and s.
   See the Language Reference Guide or a C manual for complete details.

   The special formats \n,\r,\t,\b,\f can be used to produce linefeed,
   carriage return, tab, backspace, and formfeed characters respectively.
   Use \\ to produce a backslash character and %% to produce the percent
   character.

   SPRINTF behaves like ANSI C with certain exceptions and extensions.
   These include:
   1. ANSI C requires an integer cast of a double argument to correctly
      use an integer conversion specifier like d. A similar conversion
      is required when using such a specifier with non-integral MATLAB
      values. Use FIX, FLOOR, CEIL or ROUND on a double argument to
      explicitly convert non-integral MATLAB values to integral values
      if you plan to use an integer conversion specifier like d.
      Otherwise, any non-integral MATLAB values will be outputted using
      the format where the integer conversion specifier letter has been
      replaced by e.
   2. The following non-standard subtype specifiers are supported for
      conversion characters o, u, x, and X.
      t    - The underlying C datatype is a float rather than an
             unsigned integer.
      b    - The underlying C datatype is a double rather than an
             unsigned integer.
      For example, to print out in hex a double value use a format like
      '%bx'.
   3. SPRINTF is "vectorized" for the case when A is nonscalar. The
```

```
   format string is recycled through the elements of A (columnwise)
   until all the elements are used up. It is then recycled in a similar
   manner through any additional array arguments.

See the reference page in the online help for other exceptions,
extensions, or platform-specific behavior.

Examples
   sprintf('%0.5g',(1+sqrt(5))/2)       1.618
   sprintf('%0.5g',1/eps)               4.5036e+15
   sprintf('%15.5f',1/eps)              4503599627370496.00000
   sprintf('%d',round(pi))              3
   sprintf('%s','hello')                hello
   sprintf('The array is %dx%d.',2,3)   The array is 2x3.
   sprintf('\n') is the line termination character on all platforms.

See also fprintf, sscanf, num2str, int2str.
```

（注意：上述代码会根据用户所安装的 Matlab 版本不同而略有差异。）

上面的例子都说明，在进行数据与字符串之间的转换时，格式声明是非常重要的。下边的表格以 pi 为例说明了不同格式声明下的具体转换结果：

命令	结果
sprintf('%.0e',pi)	3e+000
sprintf('%.1e',pi)	3.1e+000
sprintf('%.3e',pi)	3.142e+000
sprintf('%.5e',pi)	3.14159e+000
sprintf('%.10e',pi)	3.1415926536e+000
sprintf('%.0f',pi)	3
sprintf('%.1f',pi)	3.1
sprintf('%.3f',pi)	3.142
sprintf('%.5f',pi)	3.14159
sprintf('%.10f',pi)	3.1415926536
sprintf('%.0g',pi)	3
sprintf('%.1g',pi)	3
sprintf('%.3g',pi)	3.14
sprintf('%.5g',pi)	3.1416
sprintf('%.10g',pi)	3.141592654
sprintf('%8.0g',pi)	3
sprintf('%8.1g',pi)	3
sprintf('%8.3g',pi)	3.14
sprintf('%8.5g',pi)	3.1416
sprintf('%8.10g',pi)	3.141592654

上表中，格式指示符 e 表示转换成指数形式，f 表示带有多少个小数位，g 表示使用 e

或 f 中较短的表达式。注意，对于 e 和 f，小数点右边的数字表示在小数点后要显示多少位，而对于 g，小数点右边的数字表示总共显示多少位。在表中最后 5 种类型中，小数点前面的数字指定了 8 个字符的宽度，其显示结果是右对齐的。在表中最后一个例子中，Matlab 忽略了指定的宽度 8，因为该命令指定要显示 10 位数字。

尽管 int2str 和 str2num 函数在进行文本显示时具有比较低的效率，但当用户需要对一个字符串中的某个数字进行转换，或者从一个字符串中提取某一数字时，这两个函数就比较实用。另外，Matlab 还提供了 sscanf 和 str2double 函数用于完成相同的操作。例如，下面的代码演示了 num2str 及其逆操作函数 str2num 的用法：

```
>> s = num2str(pi*eye(2)) % create string data
s =
3.1416          0
     0     3.1416

>> ischar(s) % True for string
ans =
     1

>> m = str2num(s) % convert string to number
m =
    3.1416          0
         0     3.1416

>> isdouble(m) % Oops, this function doesn't exist
??? Undefined function or variable 'isdouble'.

>> isnumeric(m) % True for numbers
ans =
     1

>> isfloat(m) % True for floating point numbers
ans =
     1

>> pi*eye(2) - m % accuracy is lost
ans =
-7.3464e-006            0
           0 -7.3464e-006
```

函数 str2num 可以对一个表达式的值进行转换，但该表达式中不能包含工作区中的变量，如下例所示：

```
>> x = pi; % create a variable

>> ss = '[sqrt(2) j; exp(1) 2*pi-x]' % string with variable x
ss =
[sqrt(2) j; exp(1) 2*pi-x]

>> str2num(ss) % conversion fails because of x
ans =
    []

>> ss = '[sqrt(2) j; exp(1) 2*pi-6]' % replace x with 6
ss =
[sqrt(2) j; exp(1) 2*pi-6]
```

```
>> str2num(ss) % now it works
ans =
1.4142      0 +         1i
2.7183      0.28319

>> class(ans)  % yes, its a double
ans =
double
```

函数 sscanf 执行与 sprintf 函数相反的操作，它根据格式指示符从一个字符串中读取数据，如下例所示：

```
>> v = version          % get MATLAB version as a string
v =
7.0.0.51483 (R14)

>> sscanf(v,'%f')       % get floating point numbers
ans =
          7
          0
    0.51483
>> sscanf(v,'%f',1)     % get just one floating point number
ans =
    7
>> sscanf(v,'%d')       % get an integer
ans =
    7
>> sscanf(v,'%s')       % get a string
ans =
7.0.0.51483(R14)
```

sscanf 也是一个功能强大、使用灵活的函数。有关该函数的更详细信息，请参看联机帮助文档。

函数 str2double 用于把一个字符串转换成一个双精度值。虽然函数 str2num 也能实现这项功能，但是 str2double 的执行速度总体上要快一些，因为它对转换范围进行了更严格的限制。下面给出了该函数的应用实例：

```
>> str2double('Inf') % It does convert infinity
ans =
    Inf

>> class(ans)
ans =
double

>> str2double('34.6 - 23.2j') % complex numbers work
ans =
        34.6-       23.2i

>> str2double('pi') % variables and expressions don't work
ans =
  NaN
```

9.3 字符串求值

有时候，将字符串视为一个表达式进行求值是很方便的，如上节讲到的 str2num 和 str2double 函数，但这两个函数只能从字符串中提取一个个的数值，无法将字符串表达式视为一个整体进行求值操作。鉴于此，Matlab 提供了函数 eval 和 evalc 对任何符合 Matlab 语法的字符串表达式进行求值操作，如下面的代码所示：

```
>> funs = char('ceil','fix','floor','round')
funs =
ceil
fix
floor
round
>> [debland(funs(1,:)) '(pi)']  % display string to evaluate
ans =
ceil(pi)
>> f = eval([deblank(funs(1,:)) '(pi)'])
f =
    4
>> class(f)  % data type of output is numeric double
ans =
double

>> fc = evalc([deblank(funs(1,:)) '(pi)']) % try evalc
fc =
ans =
    4
>> class(fc)  % output of evalc is a character string
ans =
char
>> size(fc)
ans =
    1   13
```

由上面的代码可以看出，函数 eval 使用 Matlab 的命令解释器来求一个字符串输入的值，并将结果返回到输出参数中。函数 evalc 同样使用 Matlab 的命令解释器来求一个字符串输入的值，但返回的是字符格式的结果。也就是说，它返回的是在命令窗口中显示的结果。

在大部分情况下，我们尽可能不提倡用户使用 eval 和 evalc 函数，因为这两个函数将调用整个 Matlab 解释器执行字符串表达式的值，这就带来的巨大的运算开销。另外，这两个函数不能被 Matlab 编译器编译。（Matlab 编译器是 Matlab 的一个附属产品，它可以将 Matlab 代码转换为可执行的 C 代码）

9.4 字符串函数

Matlab 提供了许多与字符串有关的函数，其中一些函数我们已经讨论过了。下表给出了这些函数的简要描述：

函数	描述
char(S1,S2, ...)	利用给定的字符串或单元数组创建字符数组
double(S)	将字符串转换成 ASCII 形式
cellstr(S)	利用给定的字符数组创建字符串单元数组
blanks(n)	生成由 n 个空格组成的字符串
deblank(S)	删除尾部的空格
eval(S),evalc(S)	使用 Matlab 解释器求字符串表达式的值
ischar(S)	判断 S 是否是字符串数组，若是，就返回 True，否则返回 False
iscellstr(C)	判断 C 是否是字符串单元数组，若是，就返回 True，否则返回 False
isletter(S)	判断 S 是否是字母，若是，就返回 True，否则返回 False
isspace(S)	判断 S 是否是空格字符，若是，就返回 True，否则返回 False
isstrprop(S, 'property')	判断 S 是否为给定的属性，若是，就返回 True，否则返回 False
strcat(S1,S2,...)	将多个字符串进行水平串连
strvcat(S1,S2,...)	将多个字符串进行垂直串连，忽略空格
strcmp(S1,S2)	判断两个字符串是否相同，若是，就返回 True，否则返回 False
strncmp(S1,S2,n)	判断两个字符串的前 n 个字符是否相同，若是，就返回 True，否则返回 False
strcmpi(S1,S2)	判断两个字符串是否相同（忽略大小写），若是，就返回 True，否则返回 False
strncmpi(S1,S2,n)	判断两个字符串的前 n 个字符是否相同（忽略大小写），若是，就返回 True，否则返回 False
strtrim(S1)	删除字符串前后的空格
findstr(S1,S2)	在一个较长的字符串中查找另一个较短的字符串
strfind(S1,S2)	在字符串 S1 中查找字符串 S2
strjust(S1,type)	按指定的方式调整一个字符串数组，type 分为左对齐、右对齐或者居中对齐
strmatch(S1,S2)	查找符合要求的字符串的下标
strrep(S1,S2,S3)	将字符串 S1 中出现的 S2 用 S3 代替
strtok(S1,D)	查找 S1 中第一个给定的分隔符之前和之后的字符串
upper(S)	将一个字符串转换成大写
lower(S)	将一个字符串转换成小写
num2str(x)	将数字转换成字符串
int2str(k)	将整数转换成字符串
mat2str(X)	将矩阵转换成字符串，供 eval 使用
str2double(S)	将字符串转换成双精度值
str2num(S)	将字符串数组转换成数值数组
sprintf(S)	创建含有格式控制的字符串
sscanf(S)	按照指定的控制格式读取字符串

下面我们再通过几个例子对上表中出现的部分函数进行说明。例如，函数 findstr 将返回一个字符串在另一字符串中出现的位置，如下所示：

```
>> b = 'Peter Piper picked a peck of pickled peppers';
>> findstr(b,' ')       % find indices of spaces
ans =
    6   12   19   21   26   29   37
>> findstr(b,'p')       % find the letter p
ans =
    9   13   22   30   38   40   41
>> find(b=='p')         % for single character searches find works too
ans =
    9   13   22   30   38   40   41
>> findstr(b,'cow')     % cow does not exist
ans =
    []
>> findstr(b,'pick')    % find the string pick
ans =
    13   30
```

注意：函数 findstr 是区分大小写的；如果没有找到匹配的字符串，findstr 就返回空矩阵；findstr 不能用于具有多个行的字符串数组。

上表中还包含了一些逻辑判断函数（以 is 开头的函数），它们的用法如下所示：

```
>> c = 'a2 : b_c'
c =
a2 : b_c
>> ischar(c)    % it is a character string
ans =
        1
>> isletter(c)  % where are the letters?
ans =
        1       0       0       0       0       1       0       1
>> isspace(c)   % where are the spaces?
ans=
        0       0       1       0       1       0       0       0

>> isstrprop(c,'wspace')    % same as above
ans =
        0       0       1       0       1       0       0       0

>> isstrprop(c,'digit')     % True for digits
ans =
        0       1       0       0       0       0       0       0
```

最后，我们演示一下表中字符串比较函数的用法。假如一个用户输入了一个字符串（可能在一个可编辑的文本用户接口（即 uicontrol 控件）中输入），这个字符串必须和一个字符串列表的某个字符串匹配或部分匹配。这时，用户可以使用函数 strmatch 检查匹配情况，如下面的例子所示：

```
>> S = char('apple','banana','peach','mango','pineapple')
S =
apple
```

```
banana
peach
mango
pineapple
>> strmatch('pe',S)        % pe is in 3rd row
ans =
      3
>> strmatch('p',S)         % p is in 3rd and 5th rows
ans=
      3
      5
>> strmatch('banana',S)    % banana is in 2nd row
ans =
      2
>> strmatch('Banana',S)    % but Banana is nowhere
ans =
      []
>> strmatch(lower('Banana'),S) % changing B to b finds banana
ans =
      2
```

9.5 字符串单元数组

对字符串数组而言，所有的行都必须有相同的列。这一限制有时会给编程带来很多麻烦，尤其当各行的非空白部分（即除了空格以外的字符）差别很大时。因此，为了提高用户使用字符串数组的灵活性，Matlab 提供了字符串单元数组。理论上讲，所有类型的数据都可以放在单元数组中，但是单元数组最常见的用法还是用来存放字符串。在 Matlab 中，单元数组是一种数据类型，它用来存放一组不同大小和类型的数据，如下例所示：

```
>> C = {'How';'about';'this for a';'cell array of strings?'}
C =
    'How'
    'about'
    'this for a'
    'cell array of strings?'
>> size(C)
ans =
    4   1
```

注意：在上面的代码中，花括号{}用来创建单元数组，字符串单元数组在显示时会显示出两边的引号。在上例中，单元数组 C 有 4 行 1 列。读者可以发现，这个单元数组的每个单元所包含的字符串长度是不相同的。

单元数组的寻址方法和其他数组相同，如下所示：

```
>> C(2:3)
ans =
       'about'
       'this for a'
```

```
>> C([4 3 2 1])
ans =
      'cell array of strings?'
      'this for a'
      'about'
      'How'
>> C(1)
ans =
     'How'
```

上面利用圆括号( )进行寻址的方式称为“按单元索引”，返回的结果还是一个单元数组。也就是说，C(indices)根据给定的单元寻址，而不是针对这些单元的内容进行寻址。为了得到一个指定单元的内容，需要使用花括号{}，请看下面的代码：

```
>> s = C{4}
s =
cell array of strings?
>> size(s)
ans =
    1   22
```

如果使用函数 deal，用户还可以同时获取多个单元的内容，如下面的代码所示：

```
>> [a,b,c,d] = deal(C{:})
a =
How

b =
about
c =
this for a
d =
cell array of strings?
```

这里，C{:}意味着用一个列表表示所有的单元，等同于下面的代码：

```
>> [a,b,c,d] = deal(C{1},C{2},C{3},C{4})
a =
How
b =
about
c =
this for a
d =
cell array of strings?
```

利用 deal 也可以获取单元数组中部分单元的内容，如下面的代码所示：

```
>> [a,b] = deal(C{2:2:4})   % get 2nd and 4th cell contents
a =
about
b =
cell array of strings?
```

我们还可以获取某一特定单元的部分内容，如下面的代码所示：

```
>> C{4}(1:10)   % 4th cell, elements 1 through 10
ans =
cell array
```

函数 char 的功能很多，它可以将一个单元数组的内容转换成一个普通的字符串数组，如下例所示：

```
>> s = char(C)
s =
How
about
this for a
cell array of strings?
>> size(s)  % result is a standard string array with blanks
ans =
       4     22
>> ss = char(C(1:2))  % naturally you can convert subsets
ss =
How
about
>> size(ss)  % result is a standard string array with blanks
ans =
        2    5
```

函数 cellstr 完成与 char 相反的转换操作，该函数在转换时会将字符串数组中的空格去掉，如下例所示：

```
>> cellstr(s)
ans =
    'How'
    'about'
    'this for a'
    'cell array of strings?'
```

使用函数 iscellstr，可以判断一个变量是否是字符串单元数组变量，如下例所示：

```
>> iscellstr(C)     % True for cell arrays of strings
ans =
    1
>> ischar(C)        % True for string arrays not cell arrays of strings
ans =
    0
>> ischar(C{3})     % Contents of 3rd cell is a string array
ans =
    1
>> iscellstr(C(3)) % but 3rd cell itself is a cell
ans =
    1
>> ischar(C(3))     % and not a string array
```

```
ans =
    0

>> class(C)          % get data type or class string
ans =
cell
>> class(s)          % get data type or class string
ans =
char
```

在 Matlab 7 中，大多数的字符串函数都既适用于字符串数组，又适用于字符串单元数组。其中用的比较多的函数包括：deblank、strcat、strcmp、strncmp、strcmpi、strncmpi、strmatch 和 strrep 等。关于单元数组的详细信息请读者参看本书第 8 章或相应的 Matlab 帮助文档。

9.6 利用正则表达式搜索

利用标准字符串函数，例如 findstr，就可以在一个字符串中查找指定的字符序列，以及替换这些字符序列。不过，有时候用户需要进行特殊的查找，例如查找重复的字符、查找所有的大写字母、查找以大写字母拼写的单词、或者查找所有的以美元表示的金额（即以美元符号开头，后面是含有小数点的十进制数）等。在进行字符串查找时，Matlab 支持具有强大功能的正则表达式（Regular Expression）。正则表达式是指在某种查找模式下，进行字符串匹配所使用的公式。正则表达式在 Unix 系统中得到了广泛应用，如 grep、awk、sed、vi 等工具函数。另外，熟悉 Perl 或其他编程语言的用户也会或多或少地用到正则表达式。在 Matlab 中，实现正则表达式的方法很容易理解，并具有许多特性，包括一些所谓的扩展正则表达式（Extended Regular Expression）。如果你对正则表达式的概念十分清楚，那么其 Matlab 实现也不在话下。如果对正则表达式的概念不十分了解，也不要紧，可以由浅入深，先从其简单的特性用起，然后再使用更为复杂的特性。

为了使读者循序渐进地了解正则表达式的概念，我们先介绍几个使用正则表达式的简单规则。我们可以创建一个字符串公式（又称为表达式），用来描述标识字符串匹配部分的准则。该表达式由一系列字符和可选的修正符组成，修正符通常用于定义子字符串匹配的准则。最简单的字符串表达式是一个完全由文字符号组成的字符串，如下面的代码所示：

```
>> str = 'Peter Piper picked a peck of pickled peppers.'; % create a string

>> regexp(str,'pe')    % return the indices of substrings matching 'pe'
ans =
    9   22   38   41
```

上例在 str 中寻找到了 4 处'pe'，分别位于 Piper、Peck 和 peppers 中（其中 peppers 中含有两个'pe'）。

我们也可以使用字符类来匹配指定类型的字符，如一个字母、一个数字或一个空格符，也可以用来匹配一个指定的字符集。最有用的字符类是一个句点（.），它用来表示任意的单个字符。另外一个有用的字符类是位于方括号（[]）之中的字符序列或某一部分字符，这一语法通常用来表示寻找与方括号中任何一个字符元素匹配的字符串子集。例如，我们

要在 str 中寻找第一个字符为 p，最后一个字符为 r 或 c 的三字符组合，可使用下面的代码：

```
>> regexpi(str,'p.[cr]')
ans =
    9   13   22   30   41
>> regexp(str,'p.[cr]','match')     % list the substring matches
ans =
    'per'    'pic'    'pec'    'pic'    'per'
```

我们也可以使用下面的代码寻找 str 中所有的大写字母：

```
>> regexp(str,'[A-Z]')      % match any uppercase letter
ans =
    1    7
```

下表给出了一些在字符串中查找单个字符的字符串表达式：

字符串表达式	描述和用法
.	用于查找任意的单个字符（包括空格符）
[abcd35]	用于查找方括号中给出的任何一个字符
[a-zA-Z]	用于查找任意的字母，包括大写字母和小写字母，其中的连接符（-）表示字符范围
[^aeiou]	用于查找非小写的任意一个元音字母，其中的 ^ 表示对集合进行取反
\s	用于查找任意的空白符，包括空格符、制表符、换页符、换行符或回车符，相当于组合表达式[\t\f\n\r]
\S	用于查找任意非空白符，相当于组合表达式[^\t\f\n\r]
\w	查找任意的文字字符，包括大写和小写的字母、数字和下划线，相当于[a-zA-Z_0-9]
\W	查找任意的非文字字符，相当于[^a-zA-Z_0-9]
\d	查找任意的数字，相当于[0-9]
\D	查找任意的非数字字符，相当于[^0-9]
\xN 或\x{N}	查找十六进制值为 N 的字符
\oN 或\o{N}	查找八进制值为 N 的字符
\a	查找告警、提示或发声字符，相当于\o007 或\x07
\b	查找退格字符，相当于\o010 或\x08
\t	查找横向制表符，相当于\o011 或\x09
\n	查找换行符，相当于\o012 或\x0A
\v	查找纵向制表符，相当于\o013 或\x0B
\f	查找换页符，相当于\o014 或\x0C
\r	查找回车符，相当于\o015 或\x0D
\e	查找退出符，相当于\o033 或\x1B
\	使用单个反斜线符号表示要查找其后面的那个字符，主要用来查找正则表达式中具有特殊意义的字符，例如，字符串表达式\.*\?\\表示要查找句点、星号、问号和反斜线符号

上表中的字符表达式可以使用正则表达式修正符或数量词进行修改。例如，下面的代码用于在 str 中查找所有的单词，其中使用到了修正符 +：

```
>> regexp(str,'\w+','match')        % find all individual words
ans =
    'Peter'    'Piper'    'picked'    'a'    'peck'    'of'
'pickled'    'peppers'
```

由上例可以看出，修正符使字符表达式的含义发生了变化：不含 + 的表达式表示查找所有的单个文字字符，而含有 + 的表达式表示查找所有的单词，即由一个或多个文字字符组成的字符组合。

Matlab 中的字符修正符如下表所示：

修正符	描述和用法
?	只查找零次或一次与此修正符前面的元素匹配的字符
*	查找零次或多次与此修正符前面的元素匹配的字符
+	查找一次或多次与此修正符前面的元素匹配的字符
{n}	查找 n 次与此修正符前面的元素匹配的字符
{n,}	至少查找 n 次与此修正符前面的元素匹配的字符
{n,m}	至少查找 n 次与此修正符前面的元素匹配的字符，但不超过 m 次

字符修正符具有"贪婪性"，即它们会匹配能够匹配的最长字符串。例如，下面的代码将会返回 str 中以 p 开头，以 p 结尾，中间为任意字符的最长字符串：

```
>> regexp(str,'p.*p','match')
ans =
    'per picked a peck of pickled pepp'
```

不过，字符修正符的上述特性可以被下表中的量词表达式改变：

量词表达式	描述和用法
q	匹配尽可能长的字符串（具有"贪婪性"）。该特性是默认特性，其中 q 代表上表中的某个修正符，例如?、*、+、{n,m}等
q+	匹配尽可能多的字符串组件，如果第一次匹配失败，无需重新检查字符串的任何部分（具有"所有性"）。其中 q 代表上表中的某个修正符，例如?、*、+、{n,m}等
q?	在扫描字符串时，匹配尽可能短的字符串（具有"懒惰性"）。其中 q 代表上表中的某个修正符，例如?、*、+、{n,m}等

下面的代码用于在 str 中查找由两个 p 包围的所有最短的字符串：

```
>> regexp(str,'p.*?p','match')
ans =
    'per p'    'peck of p'    'pep'
```

注意：在上面的代码中没有字符串'picked a p'。这是因为懒惰性促使 Matlab 从前向后查找字符串，当发现匹配的字符串后，继续从下一个字符开始向后查找，之前已查找过的字符将不会被重新查找。

用户可以使用圆括号将多个模式组合起来进行复合查找。例如，下面的代码用于在 str 中查找以 p 开头，而不以 i 结尾的所有最长的字符串：

```
>> regexp(str,'p[^i]+','match')
ans =
    'per p'     'peck of p'     'peppers'
```

上面的代码中使用了修正符+。如果在上面的代码中加入圆括号，则表达式的意义将会发生细微的变化。例如，下面的代码在查找时，每次都会同时查找两个字符，将返回所有由双字符 p× （×为不是 i 的所有字符）构成的最长的字符串：

```
>> regexp(str,'(p[^i])+','match')
ans =
    'pe'    'pe'    'pepp'
```

由于我们在表达式中加入了圆括号，所以修正符+将作用于括号内的两个字符。

我们也可以再利用正则表达式在字符串中进行条件查找。Matlab 提供了两种条件查找所用的操作符：逻辑操作符和范围操作符。范围操作符通常为查找限定了如下条件：被查找的字符串之后（或之前）具有（或不具有）另一个匹配字符串。例如，下面的代码用于在 str 中查找所有以 d 结尾的单词之后的单词：

```
>> regexp(str,'(?<=d\s)\w+','match')
ans =
    'a'     'peppers'
```

上面的代码实际上返回的是位于字符 d 之后，到下一个空格符之前的那部分字符序列。

下表给出了 Matlab 提供的范围操作符：

范围操作符	描述和用法
p(?=q)	向前查找，查找所有符合模式 q 的字符串之前的符合模式 p 的字符串
p(?!q)	向前查找，查找所有不符合模式 q 的字符串之前的符合模式 p 的字符串
(?<=q)p	向后查找，查找所有符合模式 q 的字符串之后的符合模式 p 的字符串
(?<!q)p	向后查找，查找所有不符合模式 q 的字符串之后的符合模式 p 的字符串

除了范围操作符外，我们还可以使用逻辑操作符进行字符串的条件查找匹配。例如，下面的代码用于在 str 中查找含有 ip 或 ck 的单词：

```
>> regexp(str,'\w*(ip|ck)\w*','match')
ans =
    'Piper'     'picked'        'peck'      'pickled'
```

下表给出了 Matlab 提供的逻辑操作符：

逻辑操作符	描述和用法
p|q	查找符合模式 p 或模式 q 的字符串
^p	查找出现在原字符串最前端，并且符合模式 p 的字符串
p$	查找出现在原字符串最后端，并且符合模式 p 的字符串
\<p	查找出现在一个单词最前端，并且符合模式 p 的字符串

（续表）

逻辑操作符	描述和用法
p\>	查找出现在一个单词最后端，并且符合模式 p 的字符串
\<p\>	查找符合模式 p 的单词字符串

我们还可以使用一些标记查找符合匹配条件的重复出现的序列。当我们在表达式中使用圆括号时，与圆括号中的表达式（称为一个标记）相匹配的字符串就会被作为一个整体存储，并可以重复使用。在一个表达式中，最多可以出现 255 个标记。标记通常采用下面的语法引用：\d。其中 d 为所引用标记的索引值。比如，\1 代表第一个标记，\2 代表第二个标记，等等。例如，下面的代码用于返回 str 中所有的由两个相同字母构成的字符串：

```
>> regexp(str,'(\w)\1','match')
ans =
    'pp'
```

我们再来看一个稍微复杂一些的代码，如下所示：

```
>> regexp(str,'(.)\w*(\s)\1\w*\2','match')
ans =
    'Peter Piper'
```

上面的代码实际上返回的是：以任意一个字符开始，后面紧跟若干个文字字符，然后紧跟一个空白符，然后再紧跟一个与开头字符相同的字符，然后再紧跟若干个文字字符，随后是一个与前面的空白符相同的空白符。注意：由于 peppers 后面没有空格符，因此字符串'pickled peppers'是不符合要求的（尽管其他条件都符合）。

下表给出了 Matlab 提供的标记表达式：

标记表达式	描述和用法
(p)	查找符合标记中表达式 p 的所有字符
(?:p)	将所有符合表达式 p 的字符组合在一起，但不保存在一个标记中
(?>p)	逐元素进行组合，但不保存在一个标记中
(?#A Comment)	在表达式中插入注释（注释在执行时将被忽略）
\N	表示与该表达式中的第 N 个标记相同（例如，\1 是第一个标记，\2 是第二个标记，等等）
$N	在一个替换字符串中插入一个与第 N 个标记相匹配的字符串（仅适用于 regexprep 函数）
(?<name>p)	查找符合标记中表达式 p 的所有字符，并将标记命名为 name
\k<name>	表示与名为 name 的标记相匹配
(?(T)p)	如果标记 T 已产生（即已成功完成标记 T 的匹配），就查找符合模式 p 的字符串。该表达式实际上是一个 IF/THEN 结构，其中的标记既可以是已命名的标记，也可以是一个位置标记
(?(T)p|q)	如果标记 T 已产生，就查找符合模式 p 的字符串，否则查找符合模式 q 的字符串。该表达式实际上是一个 IF/THEN/ELSE 结构，其中的标记既可以是已命名的标记，也可以是一个位置标记

在 Matlab 中，有 3 种不同类型的正则表达式：第一种是前面使用的 regexp；第二种是 regexpi，该函数在查找时忽略大小写；第三种是 regexprep，该函数使用正则表达式替换字符串。

例如，下面的代码查找以'pi'开头的单词，并将这些单词保存起来，最后返回得到的字符串结果：

```
>> regexprep(str,'(pi\w*)(.*)(pi\w*)','$3$2$1')
ans =
    'Peter Piper pickled a peck of picked peppers'
```

上例产生了 3 个标记：标记 1 是字符串'picked'，标记 2 是字符串'a peck of'，标记 3 是字符串'pickled'。替换字符串'$3$2$1'告诉 regexprep 函数去掉原始字符串中与第一个参数相匹配的字符串，并使用与替换字符串标记相匹配的字符串来替换去掉的字符串。原始字符串剩余部分将原封不动地返回。上例中，查找模式中的数字标记使用 \N 语法引用，而替换字符串中的标记则使用 $n 语法引用。

下面的表格给出了 Matlab 中的正则表达式函数：

函数	描述和用法
regexp	使用正则表达式查找子字符串
regexpi	使用正则表达式查找子字符串，忽略大小写
regexprep	使用正则表达式查找并替换子字符串

我们在使用和修改上述正则表达式函数时也可以使用多个选项。其中一个选项是我们在前面已经用过的'match'参数。在通常情况下，regexprep 是区分大小的，并且将替换它所查找到的所有的子字符串，用户可以使用选项来改变该函数的这一默认设置。

以上 3 个函数既可以用于字符串单元数组，也可以用于单个字符串。用户既可以利用一个模式查找多个字符串，也可以利用多个模式查找多个字符串。在编程时，这些函数的部分或全部的输入参数均可以使用字符串单元数来表示。有关这些函数的更详尽的信息，请参考相应的 Matlab 帮助文档。

Chapter 10

关系和逻辑运算

与其他软件编程语言（如 C 语言、Basic 语言等）一样，Matlab 也支持关系和逻辑运算。这种运算主要是为了解决用户编程过程中的“真/假”问题。在 Matlab 中，关系和逻辑运算应用非常普遍，尤其在 M 脚本文件中，当用户进行流程控制和确定指令执行顺序时，往往需要利用关系和逻辑运算的结果（True/False）提供正确的控制信息。

> Matlab 在对一个确定的数组执行关系和逻辑运算时，把所有的非 0 数值视为 true，把 0 视为 false；在对一个表达式执行关系和逻辑运算时，如果表达式的值为真，就返回 True(1)，如果为假，就返回 False(0)。

逻辑数组是一类特殊的数组，该数组的每个元素均由 True 或 False 组成，并占据 1 个字节的存储空间。逻辑数组主要用于数组寻址（有些内容已在第 5 章涉及到），也可用于数值型表达式的运算。

10.1 关系运算符

Matlab 的关系运算包括所有我们常用的比较运算，如下表所示：

关系运算符	描述
<	小于
<=	小于或等于
>	大于
>=	大于或等于
==	等于（请不要和赋值等号=混淆）
~=	不等于

Matlab 的关系运算主要用来逐元素比较两个同维数组的大小，或者将一个数组中的每个元素与一个标量进行比较，结果都返回一个与原来的数组同维的逻辑数组。注意：当将一个数组与一个标量进行比较时，将首先利用标量扩展将标量扩展成与数组相同大小的一个数组，然后再进行逐元素的比较，从而得到的原数组相同维数的一个逻辑数组。

为了介绍上表中关系运算符的用法，我们首先生成两个一维数组（向量）A 和 B，代码如下：

```
>> A = 1:9, B = 9-A
A =
    1    2    3    4    5    6    7    8    9
B =
    8    7    6    5    4    3    2    1    0
```

下面，我们将 A 与标量 4 进行比较，看 A 的哪些元素大于 4，哪些元素不大于 4，代码如下：

```
>> tf = A>4
tf =
    0    0    0    0    1    1    1    1    1
```

从结果可以看到，在 A 中，大于 4 的元素对应的位置返回了 1（True），而不大于 4 的位置返回了 0（False)。下面，我们再验证一下 A 与 B 的对应元素是否相等，代码如下：

```
>> tf = (A==B)
tf =
    0    0    0    0    0    0    0    0    0
```

由于 A 与 B 没有一个对应元素相等，因此上述代码返回一个全 False（0）向量。

> 请读者一定要区分开“=”和“==”两个不同的概念：“==”执行的是关系运算，指将两个变量进行比较，它返回的是逻辑值，即相等时返回 True（1），不相等时返回 False（0）；“=”执行的是赋值运算，它将等号右边的变量或表达式的值赋给左边的变量。

有一点请读者注意：当在两个浮点数之间进行比较运算时，有时候会产生令人意想不到的结果。我们先看下面的例子：

```
>> tf = (-0.08 + 0.5 -0.42)==(0.5 - 0.42 - 0.08) % equal ?
tf =
    0
>> tf = (-0.08 + 0.5 -0.42)~=(0.5 - 0.42 - 0.08) % not equal ?
tf =
    1
>> (-0.08 + 0.5 -0.42)-(0.5 - 0.42 - 0.08) % not exactly equal!
ans=
    -1.3878e-017
```

从数学角度，我们一眼就能发现(−0.08 + 0.5 −0.42)和(0.5 − 0.42 − 0.08)应该是相等的，但 Matlab 却给出了相反的结果，这是为什么？这里面的主要原因在于 Matlab 在进行浮点运算时的有限精度效应，正是有它的存在，导致了有时候数学运算并不总满足交换律，因此，上述代码才得出了令用户费解的结果。有限精度效应一直是 Matlab 难以解决的问题，是 Matlab 开发商明确指出的一个 bug。关于有限精度效应的更多内容，请大家参考本书第 2 章或相关的帮助文档。

关系表达式可以和数学运算表达式进行混合运算。例如，下面的代码先将 A 与标量 2 进行比较，然后从数组 B 中减去这个比较结果：

```
>> tf = B - (A>2)
tf =
     8     7     5     4     3     2     1     0    -1
```

从结果可以看出，关系表达式和数学运算表达式进行混合运算时，逻辑数组中的 True（1）被视为数值 1，False（0）被视为数值 0。下面这条语句是关系运算的一个典型应用，它将数组 B 中的 0 元素用 eps 值（约为 2.2×10^{-16}）代替：

```
>> B = B + (B==0)*eps
B =
    Columns 1 through 7
        8.0000  7.0000  6.0000  5.0000  4.0000  3.0000  2.0000
    Columns 8 through 9
        1.0000  0.0000
```

上面这条语句的重要作用是，当将 B 中的元素作除数时，可以避免除数为 0 的情况。

有 Matlab 编程经验的用户都知道，除数为 0 是 Matlab 调试和分析过程中经常遇到的问题，对该问题的妥善解决，可以使程序开发事半功倍。下面的代码给出了一个除数为 0 的例子：

```
>> x = (-3:3)/3
x =
    -1.0000     -0.6667     -0.3333     0    0.3333  0.6667  1.0000
>> sin(x)./x
Warning:Divide by zero.
ans =
    0.8415  0.9276  0.9816  NaN 0.9816  0.9276  0.8415
```

由于 x 的第四个元素为 0，因此当将 x 作为除数进行除法运算时，Matlab 就给出了一条告警信息，告诉用户出现了被 0 除的情况，并在除数为 0 的位置给出结果 NaN（意思是 Not-a-Number——非数值）。

如果我们使用关系运算将 x 中的 0 用 eps 代替，问题就可以迎刃而解。下面给出了解决的代码：

```
>> x = x + (x==0)*eps;
>> sin(x)./x
ans =
    0.8415  0.9276  0.9816  1.0000  0.9816  0.9276  0.8415
```

我们看到，结果中的第四个元素不再是 NaN，而是得出了 sin(x)/x 当 x＝0 时的正确极限值 1。另外，用户也可以使用逻辑数组寻址的方法解决除数为 0 的问题，如下面的代码所示：

```
>> x = (-3:3)/3 % recreate x
x=
    Columns 1 through 6
        -1       -0.66667    -0.33333       0        0.33333      0.66667
```

```
    Column 7
         1
>> y = ones(size(x)) % create default output
y =
         1      1      1      1      1      1      1
>> tf= x~=0  % find nonzero locations
tf =
         1      1      1      0      1      1      1
>> y(tf) = sin(x(tf))./x(tf) % operate only on nonzeros
y =
    Columns 1 through 6
      0.84147   0.92755    0.98158     1      0.98158     0.92755
    Column 7
      0.84147
```

上面的代码虽然看上去要比前一种方法稍微麻烦一些，但却可以获得异曲同工之妙。

10.2 逻辑运算符

逻辑运算就是我们通常说的“与、或、非”，主要用于将多个关系表达式组合在一起，或对关系表达式进行取反操作，包括下表中的 5 个逻辑运算符：

逻辑运算符	描述
&	在两个逻辑数组之间进行逐元素的与操作
|	在两个逻辑数组之间进行逐元素的或操作
~	对一个逻辑数组进行取反操作
&&	标量关系表达式的避绕式（Short-Circuiting）与操作
||	标量关系表达式的避绕式（Short-Circuiting）或操作

下面，我们仍用上节创建的数组 A 和 B 来演示这些逻辑运算符的用法。首先利用关系运算创建一个逻辑数组 tf，代码如下：

```
>> A = 1:9; B=9-A; % recall data
>> tf = A>4
tf =
    0      0      0      0      1      1      1      1      1
```

我们可以利用逻辑非（~）运算符将 tf 中的元素取反，代码如下：

```
>> tf = ~(A>4)
tf =
    1      1      1      1      0      0      0      0      0
```

下面的代码在两个关系表达式之间执行逻辑与（&）操作。

```
>> tf = (A>2) & (A<6)
tf =
    0      0      1      1      1      0      0      0      0
```

从结果可以看出，上述语句找出 A 中大于 2 小于 6 的元素，并在相应的位置返回 True（1）。

标量避绕式逻辑操作符（&&和||）是 Matlab7 新增的两个逻辑操作符，“标量”指这两个操作符只用于标量关系表达式中，“避绕式”指 Matlab 按顺序执行由这两个操作符连接的标量关系表达式，当执行到某一表达式时就已经知道该逻辑运算的结果，则就不再执行后面的表达式，也就是说将绕过后面的表达式直接得出逻辑结果。为了更好地理解“避绕式”的含义，请看下面的示例代码：

```
>> a=0; b=pi;  % new data
>> a==0 || b~=1  % a=0 so this is True and b~=1 is not evaluated
ans =
     1
>> b==1 && a==0  % b~=1 so this is False and a==0 is not evaluated
ans =
     0
>> a==0 || (1/a)<1  % a==0 so 1/a is not computed
ans =
     1
```

上述最后一条语句充分说明了“避绕”的存在：由于 a==0 已经为 True，这样整个或操作的结果也必将为 True，因此，Matlab 就不再执行(1/a)<1 这一关系操作，假如 Matlab 执行这一操作的话，必将会给出一个告警信息，说明用 0 作了除数（因为 a=0）。

10.3　运算符优先级

Matlab 在执行含有关系运算和逻辑运算的数学运算时，同样遵循一套优先级原则。原则规定：Matlab 首先执行具有较高优先级的运算，然后执行具有较低优先级的运算；如果两个运算的优先级相同，则按从左到右的顺序执行。下表给出了 Matlab 中各运算符的优先级顺序：

运算符	优先级
圆括号()	最高
转置（.'），共轭转置（'），乘方（.^），矩阵乘方（^）	↓
一元加法（+），一元减法（-），取反（~）	↓
乘法（.*），矩阵乘法（*），右除（./），左除（.\），矩阵右除（/），矩阵左除（\）	↓
加法（+），减法（-），逻辑非（~）	↓
冒号运算符（:）	↓
小于（<），小于等于（<=），大于（>），大于等于（>=），等于（==），不等于（~=）	↓
逐元素逻辑与（&）	↓
逐元素逻辑或（|）	↓
“避绕式”逻辑与（&&）	↓
“避绕式”逻辑或（||）	最低

用过其他编程语言的用户可以看到，上表的优先级顺序与大多数计算机编程语言基本相似。用户可以使用括号任意改变上表的运算优先级顺序，但同一级括号内部仍遵循上表的优先级顺序和原则。

Matlab 6 和 Matlab 7 对部分操作符的运算优先级进行了调整，例如，在 Matlab 5 及以前的版本中，逻辑与（&）和逻辑或（|）具有相同的优先级，但在 Matlab 6 和 Matlab 7 中，逻辑与（&）的优先级高于逻辑或（|）。另外，从上面的表格可以看出，Matlab 7 新增加的两个逻辑操作符（&&和||）具有最低的优先级。读者可以利用下面的代码验证逻辑与（&）和逻辑或（|）的优先级：

```
>> 3|2&0
ans =
      1
>> 3|(2&0)
ans =
      1
>> (3|2)&0
ans =
      0
```

如果可以的话，读者也可以将上述代码输入到 Matlab 5（或更早的版本）中，验证其结果与在 Matlab 7 中有何不同。

10.4 关系和逻辑函数

除了前面提到的关系运算符和逻辑运算符之外，Matlab 还提供了几个函数进行关系和逻辑运算，下表给出了这些函数的名称和功能描述：

函数	描述
xor(x,y)	逻辑异或函数，当 x 和 y 一个为 0，一个不为 0 时返回 True，当 x 和 y 同时为 0 或同时为 1 时返回 False
any(x)	如果 x 是向量，则当 x 的任意一个元素不为 0 时返回 True，否则返回 False；如果 x 是数组，则对于 x 的每一列，如果有一个元素不为 0，就在该列返回 True，否则返回 False
all(x)	如果 x 是向量，则只有当 x 中所有元素都不为 0 时返回 True，否则返回 False；如果 x 是数组，则对于 x 的每一列，只有当该列所有元素都为 0 时，才在该列返回 True，否则返回 False

除了这些函数之外，Matlab 还提供了一些函数用于检验某个特定的值是否存在或者某一条件是否成立，并返回相应的逻辑结果。由于这些函数大都以“is”开头，因此它们又称为“is 族”函数，下表给出了这些函数的名称和功能描述：

函数	描述
ispc	检测用户所使用的 Matlab 版本，如果是 PC（Windows）版本，该函数返回 True，否则返回 False
isstudent	检测用户所使用的 Matlab 版本，如果是学生版本，该函数返回 True，否则返回 False

（续表）

函数	描述
isunix	检测用户所使用的 Matlab 版本，如果是 UNIX 版本，该函数返回 True，否则返回 False
ismember	检测一个值或变量是否是某个集合中的元素，如果是，就返回 True，否则返回 False
isglobal	检测一个变量是否是全局变量，如果是，就返回 True，否则返回 False
mislocked	检测一个 M 文件是否被锁定（即不能被清除），如果是，就返回 True，否则返回 False
isempty	检测一个矩阵是否为空矩阵，如果是，就返回 True，否则返回 False
isequal	检测两个数组是否相等，如果相等，就返回 True，否则返回 False
isequalwithequalnans	检测两个数组是否相等，如果相等，就返回 True，否则返回 False。若数组中含有 NaN 值，也认为所有的 NaN 值是相等的
isfinite	检测数组中的各元素是否为有限值，该函数在有限值的位置返回 True，在非有限值的位置返回 False
isfloatpt	检测数组中的各元素是否为浮点值，该函数在浮点值的位置返回 True，在非浮点值的位置返回 False
isscalar	检测一个变量是否为标量，若是，返回 True，否则返回 False
isinf	检测数组中的各元素是否为无穷大值，该函数在无穷大值的位置返回 True，在有限值的位置返回 False
islogical	检测一个数组是否为逻辑数组，若是，返回 True，否则返回 False
isnan	检测一个数是否为非数值（NaN 值），若是，返回 True，否则返回 False
isnumeric	检测一个数组是否为数值型数组，如果是，返回 True，否则返回 False
isreal	检测一个数组是否为实数数组，如果是，返回 True，否则返回 False
isprime	检测一个数是否为素数，如果是，返回 True，否则返回 False
issorted	检测一个数组是否按顺序排列，如果是，返回 True，否则返回 False
automesh	如果输入参数是不同方向的向量，就返回 True，否则返回 False
inpolygon	检测一个点是否位于一个多边形区域内，如果是，返回 True，否则返回 False
isvarname	检测一个变量名是否是一个合法的变量名，如果是，返回 True，否则返回 False
iskeyword	检测一个变量名是否是 Matlab 的关键词或保留字，如果是，返回 True
issparse	检测一个矩阵是否为稀疏矩阵，如果是，返回 True，否则返回 False
isvector	检测一个数组是否为一个向量，如果是，返回 True，否则返回 False
isappdata	检测应用程序定义的数据是否存在，如果存在，返回 True，否则返回 False
ishandle	检测是否为图形句柄，如果是，返回 True，否则返回 False
ishold	检测一个图形的 Hold 状态是否为 On，如果是，返回 True，否则返回 False
figflag	检测一个图形是否是当前屏幕上显示的图形，如果是，返回 True，否则返回 False
iscellstr	检测一个数组是否为字符串单元数组，如果是，返回 True，否则返回 False

（续表）

函数	描述
ischar	检测一个数组是否为字符串数组，如果是，返回 True，否则返回 False
isletter	检测一个字符是否是英文字母（包括大、小写），如果是，返回 True，否则返回 False
isspace	检测一个字符是否是空格字符，如果是，返回 True，否则返回 False
isa	检测一个对象是否是指定的类型，如果是，返回 True，否则返回 False
iscell	检测一个数组是否为单元数组，如果是，返回 True，否则返回 False
isfield	检测一个名称是否是结构体中的域，如果是，返回 True，否则返回 False
isjava	检测一个数组是否为 Java 对象数组，如果是，返回 True，否则返回 False
isobject	检测一个名称是否为一个对象，如果是，返回 True，否则返回 False
isstruct	检测一个名称是否为一个结构体，如果是，返回 True，否则返回 False
isvalid	检测一个对象是否是可以连接到硬件的串行端口对象，如果是，返回 True，否则返回 False

10.5 NaNs 和空数组

NaNs（非数值）和空数组（[ ]）是 Matlab 中的两类特殊的数据，在进行数学运算时，它们通常都要进行特殊处理，尤其是在逻辑或者关系运算过程中。根据 IEEE 数学标准，对 NaNs 进行运算的结果仍然是 NaNs。我们先看下边这个例子：

```
>> a = [1 2 nan inf nan]    % note that NaN is not case-sensitive
a =
     1        2        NaN      Inf      NaN

>> b = 2*a
b =
     2        4        NaN      Inf      NaN
>> c = sqrt(a)
c =
     1.0000      1.4142      NaN      Inf      NaN
>> d = (a==nan)
d =
     0    0    0    0    0
>> f = (a~=nan)
f =
     1    1    1    1    1
```

上例中，第一条语句生成了一个含有 NaNs 的向量 a，第二和第三条语句分别对 a 进行乘法和开方运算，从运算结果可以看到，对 NaNs 进行数学运算的结果仍然是 NaNs。第四和第五条语句的结果或许有些出乎读者的意料：a==nan 得到的结果全部都是 0（False），而 a~=nan 却得到了全部是 1（True）的结果！这一结果表明，在 Matlab 中，不同的 NaNs 之间是不相等的。鉴于 NaNs 的这种特性，我们在进行关系运算时，就必须确定数组中是否

含有NaNs，幸好，Matlab为我们提供了一个内置函数isnan，用来寻找数组中是否含有NaNs。下面的代码便利用isnan函数寻找a中的NaNs：

```
>> g = isnan(a)
g =
    0    0    1    0    1
```

由结果可知，isnan函数在数组中NaNs的位置返回1（True）。另外，将isnan和find函数联合使用可以寻找数组中NaNs的位置索引。例如，下面的代码找到数组a中NaNs的位置，然后在这些位置上用0替代NaNs：

```
>> i = find(isnan(a))      % find indices of NaNs
i =
    3    5
>> a(i) = zeros(size(i))   % changes NaNs in a to zeros
a =
    1    2    0    Inf     0
```

空数组是开发人员自己定义的一个数据类型（这一点与NaNs不同，因为NaNs是由IEEE标准定义的），它指有一维或多维的长度为0的数组变量。空数组的表达方式也很多，最简单的一种是直接用两个方括号表示的数组，即[ ]数组。例如，下面的代码给出了几种空数组的创建方式，并用size或length查看了它们的维数或长度：

```
>> size([])          % simplest empty array
ans=
       0    0
>> c = zeros(0,5)    % how about an empty array with multiple columns!
c =
    Empty matrix: 0-by-5
>> size(c)
ans =
      0    5
>> d = ones(4,0)     % an empty array with multiple rows!
d =
   Empty matrix: 4-by-0
>> size(d)
ans =
     4    0
>> length(d)         % it's length is zero even though it has 4 rows
ans =
     0
```

使一个数组的维数为0也许会让读者感到困惑，但这在许多运算中是非常有用的，随着本书的深入，读者会对其有更深的了解。

空数组有时也出现在一些函数的返回参数中。在Matlab中，很多函数在无法返回适当结果时，往往会返回空数组。其中最典型的一个函数就是find函数，下面给出了一个具体的例子：

```
>> x = -2:2     % new data
x =
```

```
    -2   -1   0   1   2
>> y = find(x>2)
y =
    Empty matrix: 1-by-0
```

在这个例子中，数组 x 中不存在大于 2 的值，因此找不到正确的索引值，于是 find 就返回一个空数组。我们可以使用 isempty 函数测试一个返回值是否是空数组，如下例所示：

```
>> isempty(y)
ans =
    1
```

由于空数组也存在维数（如前面创建的 c 为 0×5 的数组），在 Matlab 7 中，不同维数的空数组之间是不能进行比较的，因此，验证一个数组是否是空数组时，最好不要使用关系运算，建议使用 isempty 函数。例如，要验证前面创建的 c 是否是空数组，只能采用 isempty 函数，如下面的代码所示：

```
>> c==[] % comparing 0-by-5 to 0-by-0 arrays produces an error
??? Error using ==> eq
Matrix dimensions must agree.

>> isempty(c) % isempty returns the desired result.
ans =
     1
```

当用户确知空数组是最简单的空数组（[ ]）时，关系运算也成立，只不过运算结果仍是空数组，如下例所示：

```
>> a=[]; % create an empty variable
>> a==[] % comparing equal size empties gives empty results
ans =
     []
```

我们也可以将一个非空数组与一个空数组进行比较，结果返回空数组，如下例所示：

```
>> b=1; % create nonempty variable
>> b==[] % comparing nonempty to empty produces an empty result.
ans =
     []
>> b~=[] % even not equal comparison produces an empty result.
ans =
     []
```

由上面的例子可以看出，对空数组执行关系运算时，要么返回一个错误信息（如在两个不同维数的空数组之间进行比较时），要么返回一个空数组（如前面的两个例子），这通常都不是我们想要的结果，因此，当有空数组出现时，建议用户使用 isempty 函数，尽量不要使用关系运算。

Chapter 11

流 程 控 制

与其他计算机编程语言一样，Matlab 也允许用户使用决策结构（Decision-Making Structure）控制命令执行的流程，这一特性称为流程控制。有软件编程经验的用户也许对流程控制并不陌生，即使你以前从未接触过控制流程，通过本章的学习，也会对流程控制有全面的理解。

流程控制的功能非常强大，它通过各种循环和迭代操作使一个或多个数组中的数据产生相互影响（例如，使数组中前面的数据对后面的数据产生影响），从而实现用户的运算目的。Matlab 共提供了 5 种流程控制结构，分别是 For 循环结构，While 循环结构，If-Else-End 结构，Switch-Case 结构和 Try-Catch 模块。在多数情况下，这些结构会包含不止一条的 Matlab 命令，因此它们通常出现在 M 文件中，很少在 Matlab 命令窗口中直接输入。

11.1 For 循环

For 循环结构根据用户设定的条件，对结构中的命令反复执行固定次数的操作。For 循环的一般格式如下：

```
for x = array
    (commands)
end
```

在上述格式中，x 称为循环变量，array 称为条件数组，(commands)为要执行的循环代码。For 循环是根据数组 array 中的列数决定其循环执行的次数的，也就是说，array 有多少列，commands 就执行多少次。For 循环每执行一次，x 就取 array 中的一列作为其值，一次执行结束后，x 就取 array 的下一列的值，直到 array 的最后一列。下面的代码给出了一个 For 循环的示例：

```
>> for n= 1:10
       x(n) = sin(n*pi/10);
   end
>> x
x =
    Columns 1 through 7
```

```
        0.3090  0.5878  0.8090  0.9511  1.0000  0.9511  0.8090
    Columns 8 through 10
        0.5878  0.3090  0.0000
```

上述语句的意思是当 n 从 1 到 10 依次递增时，反复执行 x(n) = sin(n*pi/10)。也就是说，首先令 n＝1，求 x(1) = sin(1*pi/10)的值；之后，令 n＝2，求 x(2) = sin(2*pi/10)的值；……；直到 n=10，求 x(10) = sin(10*pi/10)的值。执行完 n=10 时的命令后，For 循环结束，然后就开始执行 end 之后的语句（上例中就是显示 x 的最终计算结果）。

For 循环是完全按照 array 中的值进行循环索引的，因此，用户可以通过改变 array 的值，来改变 For 循环的索引顺序。例如，下面的代码按照从 10 到 1 的顺序求 x(n) = sin(n*pi/10)的值：

```
>> for n = 10:-1:1          % decrementing loop
       x(n) = sin(n*pi/10);
   end
>> x
x =
   Columns 1 through 6
       0.30902      0.58779      0.80902      0.95106      1      0.95106
   Columns 7 through 10
       0.80902      0.58779      0.30902      1.2246e-016
```

上例中的 10:-1:1 语句将创建一个从 10 到 1 依次递减的数组。从上面的结果可以看出，虽然 For 循环在执行时是按照 10 到 1 的顺序进行的，但 x 最终的结果仍以 1 到 10 的顺序显示。

只有当 array 为多列的数组时，For 循环才反复执行多次，当 array 只有一列（包括标量）时，For 循环将只执行一次，请看下面的代码：

```
>> i = 0;   % count loop iterations
>> for n = (1:10) '% right hand side is a column
       i = i+1;
       x(n) = sin(n*pi/10);
   end
>> i    % Only one time through the loop!
i =
    1
>> x
x =
   Columns 1 through 6
       0.30902      0.58779      0.80902      0.95106      1      0.95106
   Columns 7 through 10
       0.80902      0.58779      0.30902      1.2246e-016
```

上例中，由于(1:10)'只有一列（但有 10 行），For 循环只执行了一次，这一点从 I 的值可以看出。在这惟一的一次执行中，n 取(1:10)'这个列向量，因此，x 最终的计算结果仍是一个向量。

用户甚至可以利用 randperm 函数将 array 设置成随机排列的数组，使 For 循环按照随机的顺序执行，但仍不会影响 x 的最终结果的顺序，具体代码如下例所示：

```
>> array = randperm(10)
array =
   8   2   10  7   4   3   6   9   5   1
>> for n = array
       x(n) = sin(n*pi/10);
   end
>> x
x =
   Columns 1 through 6
       0.30902     0.58779     0.80902     0.95106     1       0.95106
   Columns 7 through 10
       0.80902     0.58779     0.30902     1.2246e-016
```

上例中，randperm(10)将产生从 1 到 10 的随机排序。

注意：For 循环是完全按照条件数组 array 中的值进行的，用户不能通过在 For 循环中给循环变量赋值来终止 For 循环。例如，尽管下面的代码将 n 设为 10，但 For 循环仍然执行 10 次：

```
>> for n = 1:10
       x(n) = sin(n*pi/10);
       n = 10;
   end
>> x
x =
   Columns 1 through 6
       0.30902     0.58779     0.80902     0.95106     1       0.95106
   Columns 7 through 10
       0.80902     0.58779     0.30902     1.2246e-016
```

在 For 循环语句中，任何合法的数组生成语句都可以作为条件数组，例如，下面的语句将一个随机产生的 4×5 的数组作为条件数组：

```
>> i =1;
>> for x = rand(4,5)
     y(i) = sum(x);
     i = i+1;
   end
>> y
y =
1.7325     2.3954     1.6326     1.7201     2.0872
```

上例中，循环变量 x 分别取随机数组的每一列作为每次循环的值。由于 rand 函数产生的是一个浮点数组，无法作为索引值，因此，上例添加了变量 I 作为索引值。

与 C 语言一样，Matlab 中的 For 循环也可以任意嵌套，如下面的代码所示：

```
>> for n= 1:5
       for m = 5:-1:1
           A(n,m) = n^2 + m^2;
       end
       disp(n)
   end
```

```
        1
        2
        3
        4
        5
>> A
A =
     2     5    10    17    26
     5     8    13    20    29
    10    13    18    25    34
    17    20    25    32    41
    26    29    34    41    50
```

读者需要注意的是，前面的一些例子仅仅用来演示 For 循环的用法，并不意味着它们是高效率的执行代码。若能用等效的数组方法就可以解决的问题，应尽量避免使用 For 循环语句，因为等效的数组方法的执行效率通常要比 For 循环快几个数量级。由于数组解决方法通常都是基于向量进行的，因此又被称为向量化解决方案；而 For 循环通常是基于标量进行的，因此又称为标量化解决方案。我们前面用 For 循环求解 sin 值的问题，如果采用向量化解决方案，可以采用下面的代码：

```
>> n = 1:10;
>> x = sin(n*pi/10)
x =
  Columns 1 through 6
    0.30902    0.58779    0.80902    0.95106    1    0.95106
  Columns 7 through 10
    0.80902    0.58779    0.30902    1.2246e-016
```

从结果可以看出，向量化解决方案除了速率要快几个数量级之外，其代码也要直观得多，并且可读性更好，需要用户键盘输入的字符也较少。

对于前面的嵌套式 For 循环，我们也可以利用向量化解决方案完成，代码如下所示：

```
>> n = 1:5;
>> m = 1:5;
>> [nn,mm] = meshgrid(n,m);

>> A = nn.^2 + mm.^2
A =
     2     5    10    17    26
     5     8    13    20    29
    10    13    18    25    34
    17    20    25    32    41
    26    29    34    41    50
```

当用户使用 For 循环（包括后面讲的 While 循环）语句时，应该为循环体内出现的变量预先分配好内存，这样就可以减少执行过程中为变量分配内存的时间，从而提高执行效率。如果用户不预先给变量分配内存，则 For 循环每执行一次都要花费时间来给变量分配一次内存，这样必然会降低执行效率。因此，为了提高效率，前面求 sin 值的 For 循环语句可以改写成如下的代码：

```
>> x = zeros(1,10); % preallocated memory for x
>> for n = 1:10
        x(n) = sin(n*pi/10);
    end
```

由于x已经预先分配了存储空间，因此，For循环每次执行时只需改变x(n)的值，不需要每次都为其分配内存。

从Matlab 6.5开始，Matlab增加了一系列改善措施来减少执行循环语句的时间，这些措施被统称为JIT加速器。目前，JIT加速器已适用于大部分的Matlab语法定义、数据类型和数组大小。对于一个循环语句，当满足下面的条件时，Matlab就会在其执行时利用JIT加速器对其进行优化：

（1）该循环语句为For循环语句。

（2）该循环仅包含下列数据类型：逻辑数组、字符串、双精度数据、低于64比特的整数数据类型。

（3）该循环的条件数组最高为三维数组。

（4）循环体内出现的变量均已在循环开始之前预先定义好。

（5）循环体内出现的变量均已预先分配好了内存，并且明确指定了内存大小和数据类型。

（6）循环体内的索引值都是循环变量，并且都是数量值，例如，for i=1:N，其中i为索引值。

（7）循环体内的函数都是Matlab的内置函数，不含有用户自定义的函数。

（8）如果循环体内包含条件语句（即后面将要讲到的if-then-else或switch-case结构），则该条件语句只包含标量比较操作。

（9）循环体内最多只包含一个赋值语句。

循环体内使用的数组维数越小，则JIT加速器对代码的优化越好。但随着数组维数的增加，无论是代码本身的计算时间还是对代码的运行管理时间都会增加，因此，JIT加速器对整个执行时间的优化性能也会降低。

下面的代码给出了一个Matlab利用JIT加速器对代码进行优化的例子：

```
N = 1e5;
% generate sin(x) at 1e5 points by using array mathematics
% this is often called a 'vectorized' solution.
x = linspace(0,2*pi,N);
y = sin(x);          % vectorized solution requires two lines
% redo above using JIT-acceleration.
i = 0;
y = zeros(1,N);      % initialize all variables within loop
x = zeros(1,N);      % and allocate all memory
for i=1:N            % scalar loop variable
    x(i) = 2*pi*(i-1)/N     % only built-in function calls
    y(i) = sin(x(i));
end
```

上例给出了求解一个sin序列的向量化和For循环两种解决方案。在Matlab 7中，由

于 JIT 加速器的存在，使得这两种解决方案的耗时基本相同。但如果在 Matlab 以前的版本中，For 循环解决方案要比向量化解决方案多耗费几个数量级的时间。另外，读者会发现，就上面的简单例子而言，采用 For 循环反而使代码变得复杂难懂，JIT 加速器的作用表现得也不十分明显。对于较复杂的问题，当用户无法直接用向量化解决方案描述时，JIT 加速才能真正发挥它的作用。

11.2 While 循环

上节讲到，利用 For 循环，用户可以对一组命令执行固定次数的循环运算，但有时用户希望执行无穷次的循环运算，这时可以使用 While 循环。

While 循环的一般格式如下：

```
while expression
    (commands)
end
```

在上述格式中，expression 称为条件表达式，(commands)为要执行的循环代码。在一般情况下，expression 的计算结果为一个标量，但也可以是一个数组表达式。当 expression 的结果为一个标量，且该标量为 True 时，(commands)就会被一直执行下去；当 expression 的结果为一个数组时，只有当数组中的所有元素均为 True 时，(commands)才会被一直执行下去。

下面的例子给出了一个利用 While 循环求解 Matlab 中相对浮点精度（eps）值的方法：

```
>> num = 0; EPS = 1;
>> while (1+EPS)>1
      EPS = EPS/2;
      num = num+1;
   end
>> num
num =
     53

>> EPS=2*EPS
EPS =
    2.2204e-16
```

在上面的例子中，我们采用了大写的 EPS，主要是为了避免与 Matlab 中的保留值 eps 相冲突（这是一个好习惯，用户在为一个变量命名时，最好不要与 Matlab 的保留字或内置函数名相同）。由第 2 章可知，eps 是这样定义的：一个能用双精度值表示的最小的数，当该数与 1 相加时，将产生一个与 1 距离最小的数。在上面的例子中，EPS 首先被赋初始值 1，num 用于记录循环的次数，在 While 循环体中，只要(1+EPS)>1 这个关系表达式的值为 True，While 循环中的命令就被反复执行。因为 EPS 被连续地除以 2，所以 EPS 最后将变得足够小以至于(EPS+1)将不再大于 1，也就是说，(1+EPS)>1 就为 False，这样 While 循环就宣告结束。实际上，从数学角度上讲，无论 EPS 被 2 除多少次，(1+EPS)的值始终都是大于 1 的，但在 Matlab 中，由于每个数都是用有限数量的位表示的，当 EPS 被 2 除多次后，

结果就会变得特别小，Matlab 无法用有限数量的位表示它，这样就会出现(1+EPS)不再大于 1 的情况。细心的读者会发现上例的最后一条语句将 EPS 乘以 2 再输出，这主要是因为在退出 While 循环之前，EPS 已经被 2 除了一次，正是这次除法，使得 EPS 足够小，才导致了 While 循环的退出。因此，真正的有效的 EPS 值应是退出循环前的 EPS 值乘以 2。

上面给出的例子是 expression 的结果为标量时的情况，当 expression 的结果为数组时，只有在该数组的所有元素都为 True 的情况下，While 循环才反复执行。这时，我们可以使用 any 函数作为 While 循环的条件表达式，即 while any(expression)。只有当 expression 的结果数组中的所有元素均为 True（大于 0）时，any 函数才返回一个标量逻辑值 True，While 循环也才能反复执行。

11.3　If-Else-End 结构

在很多情况下，我们需要根据某一条件来执行相应的命令。Matlab 提供了几种 If-Else-End 结构来完成这一操作。其中最简单的一种 If-Else-End 结构的格式如下：

```
if expression
    (commands)
end
```

其中，expression 为条件表达式，(commands)为要执行的命令。只有当 expression 结果中的所有元素都为 True 时，(commands)才被执行。应该注意的是，If-Else-End 结构中的语句只被执行一次，不是循环执行的。

> 当 expression 包含多个逻辑子表达式时，Matlab 将采用前一节讲到的“避绕法”计算各表达式的值。例如，如果 expression 为(expression1 | expression2)，那么只有在 expression1 为 False 的情况下才计算 expression2；类似地，如果 expression 为(expression1 & expression2)，如果 expression1 为 False，就不再计算 expression2。注意：在 If-Else-End 结构中，即使 expression 中没有使用“避绕式”操作符（&&和| |），而是使用了普通的逻辑操作符（&和|），Matlab 同样采用“避绕法”计算各表达式的值。

下面的代码利用 If-Else-End 结构给出了一个销售打折的示例：

```
>> apples =10;          % number of apples
>> cost = apples*25     % cost of apples
cost =
    250

>> if apples>5          % give 20% discount for larger purchases
        cost = (1-20/100)*cost;
    end
>> cost
cost =
    200
```

有时在 If-Else-End 结构中，用户希望在 expression 为 True 和 False 两种条件下执行不

同的操作，这时可以使用如下格式的 If-Else-End 结构：

```
if expression
    (commands evaluated if True)
else
    (commands evaluated if False)
end
```

当 expression 为 True 时，执行第一个命令集；当 expression 为 False 时，执行第二个命令集。

当用户需要根据多个条件执行多个不同的操作时，可以采用下面的 If-Else-End 结构：

```
if expression1
    (commands evaluated if expression1 is True)
elseif expression2
    (commands evaluated if expression2 is True)
elseif expression3
    (commands evaluated if expression3 is True)
elseif expression4
    (commands evaluated if expression4 is True)
elseif expression5
    .
    .
    .
else
    (commands evaluated if no other expression is True)
end
```

在上面的结构中，Matlab 将从上到下检测各个表达式，执行与所遇到的第一个为 True 的表达式相对应的命令集，并且将该表达式后面的所有语句跳过，直接退出 If-Else-End 结构。

If-Else-End 结构的一个重要作用是中断 For 循环和 While 循环的执行，这时要用到 break 命令。例如，下面的代码利用 If-Else-End 结构和 break 命令给出了 eps 值的另一种计算方法：

```
>> EPS = 1;
>> for num = 1:1000
        EPS = EPS/2;
        if (1+EPS)<=1
            EPS = EPS*2
            break
        end
    end
EPS =
    2.2204e-16

>> num
num =
    53
```

上例中，首先设置一个 For 循环使 EPS=EPS/2;语句循环执行，然后设置一个 If-Else-End

结构用来检测 EPS 是否已经足够小。如果 EPS 已经足够小，EPS 就乘以 2，然后用 break 命令强制退出 For 循环。从执行结果可知，该方法与前面的方法得到的 EPS 结果以及执行次数都完全一样。

break 是 Matlab 中的强制跳转命令，通常用在 For 循环或 While 循环中，当执行这个命令后，Matlab 就跳出 For 循环或 While 循环，转到循环之外的下一条语句。注意，break 命令只能迫使程序跳出一个循环体，也就是说，如果一条 break 语句出现在一个嵌套的 For 循环或者 While 循环结构中，Matlab 只跳出 break 语句所在的那个循环体，并不跳出整个循环结构（除非 break 在最外层的循环体内）。

除了 break 外，在 For 循环和 While 循环中应用比较多的还有 continue 命令。continue 也是一个强制跳转命令，当 Matlab 遇到 continue 语句时，就立即跳到 For 循环或者 While 循环中的 end 语句处，跳过介于 continue 命令和 end 语句之间的所有命令。continue 与 break 不一样，它不跳出循环体，而是结束一次循环，使程序执行跳转到下一次表达式检测的位置，直接执行下一次循环操作。下面的代码将前面例子中的 break 用 continue 代替，得到了同样的结果：

```
>> EPS = 1;
>> for num = 1:1000
        EPS = EPS/2;
        if (1+EPS)>1
            continue
        end
        EPS = EPS*2
        break
    end
EPS =
    2.2204e-016
```

请注意，在这里 continue 命令对 If-End 结构没有任何影响。

11.4 Switch-Case 结构

与前面讲到的 If-Else-End 结构一样，Switch-Case 结构也是一种条件分支结构，只不过该结构根据一个公共参数的不同取值来执行不同的命令组。Switch-Case 结构的一般格式如下：

```
switch expression
    case test_expression1
        (commands1)
    case{test_expression2, test_expression3, test_expression4}
        (commands2)
    otherwise
        (commands3)
end
```

在上面的语法定义中，expression 必须是一个标量或者一个字符串。case 语句实际上执行的是一个比较操作，如果 expression 是一个标量，则 case 语句实际上执行的是

expression==test_expressionN（N=1，2，3，4，…）这条语句；如果 expression 是一个字符串，case 语句实际上执行的是 strcmp(expression,test_expressionN)（N=1，2，3，4，…）这条语句。Matlab 在执行 Switch-Case 结构语句时，也是按照从上到下的顺序进行的，即 Matlab 先判断 expression 与 test_expression1 是否相等，如果相等，(command1)就被执行，后面的语句都被跳过；如果不等，就将 expression 和 test_expression2、test_expression3、test_expression4 进行比较，如果这些表达式中的任何一个等于 expression，(commands2)就被执行，并且后面的所有命令都被跳过；如果在 case 语句中没有一个表达式与 expression 相等，就执行 otherwise 语句之后的(command3)命令组。为了描述上的完整性，第二个 case 语句给出了一个以单元数组表示的表达式组，当然，这里也可以是单个表达式。另外，case 语句也可以不止两个，可以是任意多个。

与 If-Else-End 结构不同，Switch-Case 结构中的 commands 必须以单个命令或命令组的形式存在，也就是说，Switch-Case 结构中的一条 case 语句只能执行一个命令或命令组（多余的命令或命令组将被跳过）。

下面利用 Switch-Case 结构给出了一个简单的单位换算的例子：

```
x = 2.7;
units = 'm';
switch units  % convert x to centimeters
    case {'inch','in'}
        y = x*2.54;
    case {'feet','ft'}
        y = x*2.54/12;
    case {'meter','m'}
        y = x/100;
    case {'millimeter','mm'}
        y = x*10;
    case {'centimeter','cm'}
        y = x;
    otherwise
        disp(['Unknown Units: ' units])
        y = nan;
end
```

由于上例中 units = 'm'，第三条 case 语句被执行，执行结果是 y=0.027。

11.5 Try-Catch 模块

Try-Catch 模块使得用户能够捕获程序执行过程中 Matlab 发现的错误，以便决定如何对错误进行响应。Try-Catch 模块的一般格式为：

```
try
    (commands1)
catch
    (commands2)
end
```

这里，在(commands1)中的所有 Matlab 命令都被执行。如果没有 Matlab 错误出现，程序控制就直接跳到 end 语句。如果出现了错误，程序控制就立即转移到 catch 语句，执行表达式(commands2)。在(commands2)中，通常会利用 lasterr 和 lasterror 函数获取错误信息，然后采取相应的措施。

下面给出了一个利用 Try-Catch 模块检测错误的例子（建议用户利用一个 M 脚本文件保存这些代码，以便于验证）：

```
x = ones(4,2);
y = 4*eye(2);
try
    z = x*y;
catch
    z = nan;
    disp('X and Y are not conformable.')
end
z
```

执行这个例子，将得到如下的结果：

```
z =
    4    4
    4    4
    4    4
    4    4
```

在这个例子中，由于 x 和 y 都符合要求，因此代码只执行了 try 模块。下面我们改变变量 y，使代码执行产生错误：

```
x = ones(4,2);
y = 4*eye(3);       % now wrong size
try
    z = x*y;
catch
    z = nan;
    disp('X and Y are not conformable.')
end
z
```

再次执行这段代码，就会在命令窗口中得到如下输出：

```
X and Y are not conformable.
z =
   NaN
```

利用 lasterr 函数可以返回代码执行遇到了何种错误：

```
>> lasterr
ans =
Error using ==> *
Inner matrix dimensions must agree.
```

另外，利用函数 lasterror 可以将更详细的错误信息返回到一个结构体中：

```
>> errstr = lasterror
errstr =
    message: [1×53 char]
    identifier: 'MATLAB: innerdim'
>> errstr.message
ans =
Error using ==> *
Inner matrix dimensions must agree.
```

上例中，结构体 errstr 的 message 域包含了 lasterr 函数的返回值（即一个表示错误的字符串），indentifier 域包含了一个错误标识符，本例中的标识符表明错误与 Matlab 的内部维数有关。

用户也可以在 catch 模块中利用 rethrow 函数将实际的错误字符串显示出来。例如，我们可以将前面的例子稍作修改，如下所示：

```
x = ones(4,2);
y = 4*eye(3);       % now wrong size
try
    z = x*y;
catch
    z = nan;
    disp('X and Y are not conformable. ')
    rethrow(lasterror) % process error as if Try-Catch did not happen.
end
z
```

执行上面的代码，可以得到如下结果：

```
X and Y are not conformable.
???Error using ==> *
Inner matrix dimensions must agree.
```

rethrow 函数将实际的错误字符串显示出来，并终止程序执行，因此程序并没有将 z 的内容显示出来。

Chapter 12

函　数



用户也许对函数并不陌生，我们前面使用过的 inv、abs、angle、sqrt 等均为函数。在 Matlab 中，一个函数其实就是一个“黑箱”，它接受用户传递给它的值，然后根据这些值计算所需要的结果，并将结果输出给用户。在函数运行时，其内部生成的中间变量都不会显示出来，也不会存储到 Matlab 工作区窗口中（当在函数中设置断点时例外），因此，用户所看见的只是输入的参数以及输出的结果。

函数能够把大量有用的数学函数或命令集集中在一个模块中，因此，它们对某些复杂问题具有很强的解决能力。Matlab 提供了 3 种结构允许用户创建自己的函数，即 M 文件函数、匿名函数和内联函数。其中，M 文件函数最常用。下例给出了一个典型的 M 文件函数，文件名为 mmempty.m：

```
function d=mmempty(a,b)
%MMEMPTY Substitute Value if Empty.
% MMEMPTY(A,B) returns A if A is not empty,
% otherwise B is returned.
%
% Example: The empty array problem in logical statements
% let a=[]; then use MMEMPTY to set default logical state
% (a==1) is [], but MMEMPTY(a,1)==1 is true
% (a==0) is [], but MMEMPTY(a,0)==0 is true
% Also:
% sum(a) is 0, but sum(MMEMPTY(a,b))=sum(b)
% prod(a) is 1, but prod(MMEMPTY(a,b))=prod(b)
%
% See also ISEMPTY, SUM, PROD, FIND
if isempty(a)
       d=b;
else
       d=a;
end
```

从上面的 M 文件函数可以看出，M 函数文件（包含 M 文件函数的文件）与 M 脚本文件很相似，它们都是以.m 作后缀的文本文件，都可以用 Matlab 文件编辑器进行编辑，并且

文件中的命令都不能直接在命令窗口中输入，调用时只要输入文件名即可。不过，M 函数文件与 M 脚本文件也有区别，主要表现在：①虽然都是.m 文件，但 M 函数文件的文件名必须与函数名相同，而 M 脚本文件没有这一限制。②M 函数文件在调用时除了需要文件名（函数名）外，还需要提供输入参数，并通过输出参数得出计算结果，而 M 脚本文件在调用时只需要提供文件名即可。③函数文件内部创建的变量在函数执行时不会在 Matlab 窗口显示，也不会存储到 Matlab 工作区中，只有函数的输出结果才存储到 Matlab 工作区中，而 M 脚本文件创建的每个变量都会保存到 Matlab 工作区中。除了以上区别外，Matlab 在文件格式上还有一些不同于 M 脚本文件的地方：首先，一个 M 函数文件的第一行都是一个以关键字 function 开头的声明行，表明该文件为一个 M 函数文件，声明行中包含了函数名称、输入参数和输出参数；另外，为了使别的用户了解该函数的功能特点和使用方法，通常需要为其加入适当的注释，M 文件函数的注释是有一定规律的，例如，紧接着 function 声明行的那行注释（如 mmempty 文件中的注释%MMEMPTY Substitute Value if Empty.）通常称为首行帮助，简称为 H1，通常用于对函数进行总体说明，当我们用 lookfor 查看该函数信息时，显示的便是这行内容，还有，介于 function 声明行与函数命令语句之间的所有注释（包括 H1）称为函数的帮助信息，主要包括函数简介、参数说明、典型示例、参考函数等，当我们用 help 或者 helpwin 命令查看函数文件时，这些内容便被显示出来。最后需要说明的是，由于 Matlab 函数通常是通过输出参数传递计算结果的，因此，大部分 Matlab 函数中都不包含 return 命令，这时 Matlab 将按顺序执行函数所有的命令，最后通过指定的输出变量给出结果，但是，如果用户在函数中使用了 return 命令，Matlab 就会在执行到 return 命令处停止，并将此时输出变量的值作为结果输出。

12.1 M 函数文件的构建规则

本节主要阐述 M 函数文件的创建必须满足的一些标准，以及具有的一些特性。我们将这些标准和特性总结如下：

（1）M 函数文件名必须和 function 声明行中的函数名（例如 mmempty）一致。实际上，当用户键入一个函数执行时，Matlab 寻找的是以这个函数的名字命名的.m 文件，而不是 function 声明语句中的函数名。

（2）在 Matlab 7 中，一个 M 函数文件名最多可包含 63 字符。当然，这只是 Matlab 允许的最大值，根据用户的操作系统不同，M 函数文件名的有效字符数有可能小于 63 字符。当一个 M 函数文件名的长度多于 63 字符时，Matlab 将忽略第 63（或者由用户操作系统决定的最大字符数）个字符之后的所有字符。因此建议用户在命名一个 M 函数文件时，使用稍长一些的名字，这样就可以为每一个 M 函数文件提供一个惟一的名称。

（3）在 Matlab 7 之前的版本中，M 函数文件名只在 Unix 操作系统平台上区分大小写，而在 Windows 操作系统平台上是不区分大小写的。但在 Matlab 7 版本中，M 函数文件名在 Windows 操作系统平台上也区分大小写。因此，为了格式统一和避免大小写错误，建议用户只使用小写字母表示 M 函数文件名。

（4）M 函数文件名的命名规则与变量名相同，即必须以一个字母开头，后面可以是任意的字母、数字和下划线的组合，空格和标点符号不能用作文件名。

（5）M 函数文件的第一行必须是以 function 关键字开头的声明语句，称为函数声明行。function 之后的语句定义了该函数的调用方式，包括函数名、输入参数和输出参数。其中输入参数和输出参数均是该函数的局部变量，也就是说，这些参数名只在函数体内部有效。函数执行时，通过输入变量获得数据，通过输出变量输出结果。

（6）介于函数声明行和第一行命令之间的若干行注释是函数的帮助文档，当使用 help 或者 helpwin 命令查看时，将显示这些内容。其中，第一行注释称为 H1 行，通常包含对函数功能的简要描述，当使用 lookfor 命令查看时，显示的便是这行内容。H1 行之后的帮助文档内容对该函数进行了详细描述，包括函数调用方法、函数所使用的算法、一些简单示例等。

（7）为了使函数名在帮助文档中一目了然，通常帮助文档中的函数名都是大写的，但在函数调用时，必须使用与函数文件名相同的小写字母。

（8）函数帮助文档之后的所有语句称为函数体。函数体根据输入参数，通过一系列运算命令得出结果，并通过输出参数输出给用户。

（9）如果 M 函数文件中包含 return 语句，则函数执行到该语句终止；如果不包含 return 语句，则执行到文件的最后一行终止。当 M 函数文件中包含嵌套函数时，每个嵌套函数都必须有一个 end 语句来终止该嵌套函数的执行。

（10）当函数执行时出现异常或错误时，用户可以通过调用 error 函数来终止函数执行并返回命令窗口。比如，当函数中某个函数或命令使用不正确时，可以使用 error 函数指出非法使用的原因或纠正办法，下例给出了一个具体应用：

```
if length(val) > 1
   error ('VAL must be a scalar.')
end
```

如果上述 error 语句被执行（注意，只有满足了条件 length(val) > 1，该语句才被执行），则函数执行立即被终止，并在命令窗口中显示字符串'VAL must be a scalar. '，同时，在该字符串之前，还将显示出现错误的文件名。另外，标示错误的字符串将会传递给 lasterror 和 lasterr 函数，以备后面调用。注意，如果给 error 函数传递一个空字符串，即 error(' ')，将不会引发任何操作。

另外，error 函数和 sprintf 函数一样，也可以在字符串中格式化显示数字变量。例如，我们可以将前面的 error 语句改成如下的形式：

```
val = zeros(1,3);
if length(val) > 1
   error ('VAL has %d elements but must be a scalar.',length(val))
end
??? VAL has 3 elements but must be a scalar.
```

上例中除了显示错误信息外，还在信息中给出了 VAL 的长度值，其中%d 说明该值是一个整数值。当 Matlab 发现错误时，除了在命令窗口显示错误信息外，还会产生一个错误

信息指示符，该指示符（identifier）可通过函数 lasterror 显示出来，下面给出了一个简单的例子：

```
>> eig(eye(2,4))
??? Error using ==> eig
Matrix must be square.
>> lasterror
ans =
        message: [1×42 char]
     identifier: 'MATLAB:square'
```

上例中的指示符说明第一条语句中的错误信息来源于 Matlab，并要求程序使用方阵。用户也可以在自己的程序中标明错误信息指示符，只要将其作为 error 函数的第一个参数即可（其他参数依次后移）。下面给出了一个具体的例子：

```
>> val = zeros(1,3);
>> msg = 'VAL has %d elements but must be a scalar.';
if length(val) > 1
   error ('MyToolbox:scalar',msg,length(val))
end
??? VAL has 3 elements but must be a scalar.
>> lasterror
ans =
         message: 'VAL has %d elements but must be a scalar.'
      identifier: 'MyToolbox:scalar'
```

该指示符说明，前面的错误信息是由 MyToolbox 中的一个函数文件产生的，并与标量有关。

（11）在 M 文件函数中也可以调用 warning 函数，对函数执行时出现的异常行为和意外情况发出告警信息。warning 函数与前面讲到的 error 函数在语法定义和使用方法上十分相似，如参数中都可以包含简单的字符串、格式化数字以及一个可选的初始信息指示符。warning 函数与 error 函数的区别在于：①当函数执行遇到 warning 函数时，不是立即返回，而是继续执行，只不过会立即在命令窗口中显示一条告警信息。②用户可以通过全局设置关闭告警功能，或关闭与某个特定指示符相关的告警功能。有关 warning 函数的其他用法请读者参考 Matlab 帮助文档。

（12）M 函数文件内部也可以调用 M 脚本文件。此时，M 脚本文件所创建的所有变量都作为函数的内部变量，不会出现在 Matlab 工作区中。

（13）在 M 函数文件内部可以创建一个或多个子函数，又称为局部函数。这些函数通常出现在主函数体的后面，同样以一个标准的函数声明语句开始，并且遵循所有的函数创建规则。

（14）子函数可以被主函数调用，也可以被同一文件内的其他子函数调用，但不能被文件之外的其他函数调用。

（15）子函数也含有像主函数那样的帮助文档，并通过以下格式查看：>>helpwin func/subfunc，其中 func 是主函数名，subfunc 是子函数名。

（16）子函数的命名规则与主函数相同，如最多 63 字符、必须以字母开头等。但建议用户在创建子函数时均以 local（也可以是用户习惯的其他符号，如 sub 等）开头，例如，

local_myfun。这样做的目的是提高主函数的可读性，不至于造成调用时的困扰。下面给出了一个包含子函数的例子，其中主函数为 mmclass，子函数为 local_getclasslist：

```
function c=mmclass(arg)
%MMCLASS MATLAB Object Class Existence.
% MMCLASS returns a cell array of Strings containing the
% names of MATLAB object classes available with this license.
%
% MMCLASS('ClassName') returns logical True(1) if the class
% having the name 'ClassName' exists with this license.
% Otherwise logical False(0) is returned.
%
% MMCLASS searches the MATLABPATH for class directories.
% Classes not on the MATLABPATH are ignored.
%
% See also CLASS, ISA, METHODS, ISOBJECT
persistent clist   % save data for future calls

if isempty(clist)  % clist contains no data, so create it
    clist=local_getclasslist;
else
if nargin==0
     c=clist;
elseif ischar(arg)
     c=~isempty(strmatch(arg,clist));
else
     error('Character String Argument Expected.')
end

function clist=local_getclasslist
%LOCAL_GETCLASSLIST Get ist of MATLAB classes
%
% LOCAL_GETCLASSLIST returns a list of all MATLAB classes
% in a cell array of strings
clist=cell(0);
cstar=[filesep '@*'];
dlist=[pathsep matlabpath];
sidx=findstr(pathsep,dlist)+1;       % path segment starting indices
eidx=[sidx(2:end)-1 length(dlist)];  % path segment ending indices

for i=1:length(sidx)-1                % Look at each path segment
   cdir=dir([dlist(sidx(i):eidx(i)) cstar]);  % dir @* on segment
   clist=[clist {cdir.name}];                 % add results to list
end
cstr=char(clist);             % convert to string array
cstr(:,1)=[];                 % eliminate initial '@'
cstr=unique(cstr,'rows');     % alphabetize and make unique
clist=cellstr(cstr);          % back to a cell array
% end of subfunction
```

(17)除了子函数之外，M 函数文件还可调用私有函数。私有 M 函数文件也是标准的.m

文件，保存在主函数所在目录的子目录中，主要包含主函数所需的一些子功能。注意，只有直接父目录下的主函数才能调用这些私有 M 函数。

（18）同子函数一样，私有 M 函数文件名也没有什么特殊要求，其命名规则与子函数相同。但同样为了显示和调用方便，建议私有 M 函数文件名都以 private 开头，例如，private_myfun。

12.2 输入和输出参数

Matlab 函数对输入和输出参数的数量没有限制，可以有任意多个输入和输出参数。这些参数通常需要遵循一定的规则，并表现出特有的属性。具体规则和属性如下：

（1）M 文件函数可以没有输入和输出参数。

（2）用户在调用 M 文件函数时可以提供少于函数定义中规定个数的输入输出参数，但不能提供多于函数定义中规定个数的输入输出参数。

（3）用户可以通过分别调用函数 nargin 和 nargout 函数来确定一个 M 函数在调用时用到了几个输入和输出参数。注意，nargin 和 nargout 是函数，不是变量，因此用户不能对它们进行赋值操作，如 nargin=nargin-1。下面的 M 函数文件 mmdigit.m 演示了 nargin 的具体用法：

```
function y=mmdigit(x,n,b,t)
%MMDIGIT Round Values to Given Significant Digits.
% MMDIGIT(X,N,B) rounds array X to N significant places in base B.
% If B is not given, B=10 is assumed.
% If X is complex the real and imaginary parts are rounded separately.
% MMDIGIT(X,N,B,'fix') uses FIX instead of ROUND.
% MMDIGIT(X,N,B,'ceil') uses CEIL instead of ROUND.
% MMDIGIT(X,N,B,'floor') uses FLOOR instead of ROUND.

if nargin<2
   error('Not enough input arguments.')
elseif nargin==2
   b=10;
   t='round';
elseif nargin==3
   t='round';
end
n=round(abs(n(1)));
if isempty(b), b=10;
else          b=round(abs(b(1)));
end
if isreal(x)
   y=abs(x)+(x==0);
   e=floor(log(y)./log(b)+1);
   p=repmat(b,size(x)).^(n-e);
   if strncmpi(t,'round',1)
      y=round(p.*x)./p;
```

```
    elseif strncmpi(t,'fix',2)
       y=fix(p.*x)./p;
    elseif strncmpi(t,'ceil',1)
       y=ceil(p.*x)./p;
    elseif strncmpi(t,'floor',2)
       y=floor(p.*x)./p;
    else
        error('Unknown rounding requested')
    end
 else % complex input
    y=complex(mmdigit(real(x),n,b,t),mmdigit(imag(x),n,b,t));
 end
```

在上述文件中，nargin 函数被用来给用户在调用该函数时没有提供的输入参数赋以默认值。

（4）前面讲到，M 函数在执行时，内部变量并不存储到 Matlab 工作区中，这些变量将被存储到一个临时建立的工作区，该工作区称为函数工作区，当函数执行结束后，该工作区连同工作区内的内部变量一同被清除。当一个函数被调用时，函数只从输入变量中读取数值，并不将其添加到函数工作区中。但是，如果在函数执行过程中改变了输入变量中的任何一个元素，该输入变量就会被添加到函数工作区中。因此，为了节省内存和提高速度，建议用户最好不要直接更改输入变量本身，而是将需要的元素从输入变量数组中提取出来，赋给一个临时变量，然后再通过该临时变量进行修改和引用，这样就可避免将整个输入变量添加到函数工作区。请注意，如果在函数定义时某对输入参数和输出参数具有相同的变量名，函数执行时该变量将会被直接添加到函数工作区中。例如，对于函数 function y=myfunction(x,y,z)，在函数执行时，变量 y 将被立即添加到 myfunction 的工作区中。

（5）如果一个函数声明了一个或者多个输出参数，但是用户在使用时又不想要输出参数，则只需在调用该函数时不给函数提供输出变量即可。另外，在函数体内部，用户也可以在函数结束之前使用 clear 函数删除不需要的输出变量。

（6）如果在函数声明行中将 varargin 作为最后一个输入参数，则函数在调用时可以接受任意个数的输入变量。在 Matlab 中，varargin 是一个预先定义的单元数组，该单元数组的第 i 个单元就是从 varargin 出现位置算起的第 i 个输入参数。例如，对于如下声明的一个函数：

```
function a=myfunction(varargin)
```

如果该函数采用 a=myfunction(x,y,z)的方式调用，则在函数内部，varargin 包含 3 个单元，其中 varargin{1}由数组 x 构成，varargin{2}由数组 y 构成，varargin{3}由数组 z 构成。类似地，如果函数采用 a=myfunction(x)的方式调用，那么 varargin 的长度为 1，并且 varargin{1}=x。由于 myfunction 的声明中使用了 varargin 作为惟一的输入参数（它必然也是最后一个参数），因此，该函数可以用不同数量的输入参数进行调用。

如果某些输入参数在任何情况下都必须出现，则可以在函数声明时将其加入到 varargin 之前，但必须保证 varargin 作为最后的参数。例如，对于如下声明的一个函数：

```
function a=myfunction(x,y,varargin)
```

如果调用方式为a=myfunction(x,y,z)，则varargin为长度为1的单元数组，并且varargin{1}=z。另外，由于x和y在函数声明时显式出现在varargin之前，因此，无论何时调用该函数，都必须提供这两个数入参数。最后，用户可以利用nargin函数返回函数调用时实际使用的输入参数的个数。

（7）与前面讲的varargin相类似，如果在函数声明行中将varargout作为最后一个输出参数，则函数在调用时可以接受任意个数的输出变量。varargout 也是一个预先定义的单元数组，该单元数组的第i个单元就是从varargout出现位置算起的第i个输出参数。例如，对于如下声明的一个函数：

```
function varargout=myfunction(x)
```

如果该函数采用[a,b]=myfucntion(x)的方式调用，则在函数内部，varargout由两个单元构成，其中varargout{1}的值赋给变量a，varargout{2}的值赋给变量b。与varargin一样，varargout的长度等于函数调用时实际使用的输出参数的个数，用户可以使用 nargout 函数返回这个值。如果某些输出参数在任何情况下都必须出现，则可以在函数声明时将其加入到varargout之前，但必须保证varargout作为最后的参数。例如：function [a,b,varargout]=myfunction(x)。该情况下varargout的属性与varargin一致，这里不再赘述。

（8）（2）中规定，用户不能提供多于函数定义中规定个数的输入输出参数。因此，Matlab提供了nargchk和nargoutchk函数分别用于对有效的输入和输出参数个数进行判断。当函数声明时定义了固定个数的输入输出参数（即不出现varargin和varargout），而用户没有正确指定参数个数，Matlab会自动报错，nargchk和nargoutchk函数作用不大；但当函数声明时定义任意个数的输入或输出参数时（即（6）和（7）的情况），这两个函数是非常有用的（因为此时Matlab通常不会自动报错）。

12.3 函数工作区

前两节讲到，Matlab函数是一个黑箱，它通过输入参数接受数据，通过输出参数将计算结果输出，在函数体内部创建的任何变量都不会在Matlab工作区中显示。其实，Matlab函数在每次执行时都会创建一个临时工作区，我们称之为函数工作区，函数内部创建的变量都会存储在该工作区中，当函数执行完毕后该工作区连同里面的变量一起被删除。在函数调用时，函数内部的变量通常通过输入和输出参数与Matlab工作区进行数据交换，除此之外，Matlab还提供了其他一些方法和技术用于函数工作区之间以及函数工作区和Matlab工作区之间的数据交换。这些技术主要包括：

（1）如果函数体内的一个变量被声明为global，该变量就可以被其他函数、Matlab工作区以及函数本身反复调用。要访问函数或Matlab工作区中定义的全局变量，必须在每个工作区中都将该变量声明为global，声明格式为：global variablename。

> 在实际编程过程中，不建议用户使用全局变量。如果一定要使用全局变量，建议用户给全局变量命名时尽量采用长一些的名称，全部使用大写字母，并且用它们

所在的M函数文件的文件名作开头，例如，MYFUN_ALPHA。这些命名规则便于用户管理全局变量，并尽量避免由于变量重名造成的意外错误。

（2）除了通过全局变量共享数据外，M函数文件还定义了另一种可共享的变量类型：persistent变量，其声明格式为：persisitent variablename，该变量又称为永久变量。永久变量与全局变量类似，但是它只能被声明该变量的函数反复调用，不能被其他函数或Matlab工作区调用。一旦一个包含永久变量的M函数文件被调用，永久变量就会保留下来，供下次调用该函数时使用。下面的函数mmclass演示了永久变量的用法：

```
function c=mmclass(arg)
%MMCLASS MATLAB Object Class Existence.
% MMCLASS returns a cell array of strings containing the
% names of MATLAB object classes available with tnis license.
%
% MMCLASS('ClassName') returns logical True (1) if the class
% having the name 'ClassName' exists with this license.
% Otherwise logical False (0) is returned.
%
% MMCLASS searches tne MATLABPATH for class directories.
% Classes not on the MATLABPATH are ignored.
%
% See also CLASS, ISA, METHODS, ISOBJECT
persistent clist   % save data for future calls

if isempty(clist)  % clist contains no data, so create it
    clist=cell(0);
    cstar=[filesep '@*'];
    dlist=[pathsep matlabpath];
    sidx=findstr(pathsep,dlist)+1;      % path segment starting indices
    eidx=[sidx(2:end)-2 length(dlist)]; % path segment ending indices
    for i=1:length(sidx)-1              % look at each path segment
        cdir=dir([dlist(sidx(i):eidx(i)) cstar]); % dir @* on segment
        clist=[clist {cdir.name}];      % add results to list
    end
    cstr=char(clist);            % convert to string array
    cstr(:,1)=[];                % eliminate initial  '@'
    cstr=unique(cstr,'rows');    % alphabetize and make unique
    cstr=cellstr(cstr);          % back to a cell array
end
if nargin==0
    c=clist;
elseif ischar(arg)
    c=~isempty(strmatch(arg,clist));
else
    error('Character String Argument Expected.')
end
```

在mmclass函数文件中，变量clist被声明为永久变量。当第一次调用mmclass函数时，该函数将创建一个空的永久变量clist，并用接下来的一个if语句为clist赋值。即使mmclass

函数第一次调用结束，clist 变量仍然存在，并保持函数结束前的值。当用户再次调用该函数时，clist 已经是一个已经存在的变量，因此，mmclass 函数将直接从第二条 if 语句开始执行。

（3）如果用户希望在其他工作区执行一个表达式，然后将结果返回到当前工作区，则可以使用 evalin 函数。evalin 函数与 eval 函数类似，只不过 evalin 函数中的表达式既可以在调用工作区（caller workspace）执行，又可以在基本工作区（base workspace）执行。上述的调用工作区指调用当前函数的工作区；基本工作区即是 Matlab 工作区。例如，命令 A=evalin('caller', 'expression')将在调用工作区中执行表达式'expression'，然后将计算结果赋给当前工作区中的变量 A；而命令 A=evalin('base', 'expression')将在基本工作区中执行表达式'expression'，并将计算结果赋给当前工作区中的变量 A。evalin 还用于捕捉程序运行中的错误，其语法格式为：evalin('workspace', 'try', 'catch')，其中'workspace'可以是'callser'，也可以是'base'，命令首先在'workspace'中执行'try'语句，当'try'语句出现错误时，就在当前工作区中执行'catch'语句。

（4）既然用户可以在另一个工作区中执行一个表达式，并将结果赋给当前工作区中的一个变量，用户也必然可以将当前工作区中的一个表达式的结果赋值给另一个工作区中的一个变量。函数 assignin 便提供了这项功能，其语法格式：assignin('workspace', 'vname',x)，其中'workspace'可以是'caller'，也可以是'base'，该命令将当前工作区中变量 x 的值赋给'workspace'工作区中的名为'vname'的变量。

（5）当在一个函数内部使用 inputname 函数时，就可以检测当该函数被调用时，调用工作区中都有哪些变量被使用（从函数名称上也可以看出，该函数将返回输入变量的名称）。例如，假设一个函数的调用方式如下：

```
>> y = myfunction(xdot,time,sqrt(2))
```

则在 myfunction 函数中输入命令 inputname(1)将返回一个字符串'xdot'，输入命令 inputname(2)将返回'time'，输入命令 inputname(3)将返回一个空数组，这是因为 sqrt(2)不是一个变量而是一个表达式，该表达式只能生成一个临时结果。

下面的函数 mmswap 演示了前面讲到的 evalin、assignin 和 inputname 等函数的用法：

```
function mmswap(x,y)
% MMSWAP Swap Two Variables.
% MMSWAP(X,Y) or MMSWAP X Y  swaps the contents of the
% variable X and Y in the workspace where it is called.
% X and Y must be variables not literals or expressions.
%
% For example: Rat=ones(3); Tar=pi; MMSWAP(Rat,Tar) or MMSWAP Rat Tar
% swaps the contents of the variables named Rat and Tar in the
% workspace where MMSWAP is called giving Rat=pi and Tar=ones(3).
if nargin~=2
   error('Two Input Arguments Required.')
end
if ischar(x) & ischar(y)   % MMSWAP X Y 'string input arguments'
   % check existence of arguments in caller
```

```
    estr=sprintf('[exist(''%s'',''var'')
    exist(''%s'',''var'')]',x,y);
    t=evalin('caller',estr);
    if all(t)                           % both x and y are valid
        xx=evalin('caller',x);          % get contents of x
        yy=evalin('caller',y);          % get contents of y
        assignin('caller',y,xx)         % assign contents of x to y
        assignin('caller',x,yy)         % assign contents of y to x

    elseif isequal(t,[0 1])             % x is not valid
        error(['Undefined Variable:  ''' x  ''''])

    elseif isequal(t,[1 0])             % y is not valid
        error(['Undefined Variable:  ''' y  ''''])

    else                                % neither is valid
        error(['Undefined Variables:  ''' x ''' and ''' y  ''''])
    end
else                            % MMSWAP(X,Y) 'numerical input arguments'
    xname=inputname(1);         % get x argument name if it exists
    yname=inputname(2);         % get x argument name if it exists
    if ~isempty(xname) & ~isempty(yname)    % both x and y are valid
       assignin('caller',xname,y)           % assign contents of y to x
       assignin('caller',yname,x)           % assign contents of x to y

    else
       error('Arguments Must be Valid Variables.')
    end
end
```

（6）当前正被执行的 M 函数文件的文件名可以通过该函数内部工作区中的一个变量 mfilename 得到。例如，对于 M 函数文件 myfunction.m，当该文件被执行时，其函数工作区中就创建一个变量 mfilename，变量值为字符串'myfunction'。另外，对于 M 脚本文件也存在这样一个变量，用于保存正在执行的脚本文件的文件名。

12.4　Matlab 的函数文件搜索路径

M 函数文件是 Matlab 给用户提供的一项基本工具，它使得用户可以对一组有用的命令进行封装，然后反复调用。M 函数文件与 M 脚本文件一样，也被存储到计算机硬盘上，Matlab 在调用某个函数时，也需要在硬盘上搜索相应的.m 文件。因此，为了提高搜索效率和执行速度，Matlab 在搜索、打开并执行 M 函数文件时也采取了一些特有的技术和手段。这些技术和手段主要包括：

（1）当 Matlab 第一次打开并执行一个 M 函数文件时，函数中的命令将被编译（compile）成内部伪码（internal pseudocode）格式存储到内存中，以提高对该函数后续调用时的执行速度。如果函数中还包含了对其他 M 函数文件和 M 脚本文件的调用，这些文件也被编译成内存中的内部伪码格式。

（2）使用函数 inmem 可以查看当函数执行时被编译成内存伪码格式的各个函数和脚

本文件名，这些文件名被保存在该函数返回的一个字符串单元数组中。

（3）当一个函数被编译到内存中后，可以使用 mlock 命令将该函数锁定在内存中，这样就保证它不会从内存中被清除，也就是说，利用 clear 命令将不会将该函数从内存中清除出去。如果一个 M 函数文件被锁定，则该函数中声明的永久变量就会驻留在内存中，以便被其他函数反复调用。与 mlock 命令相对应的命令是 munlock，它将一个锁定的函数解锁，使其可以从内存中清除。函数 mislocked 用于检查一个函数是否被锁定，如果函数被锁定，就返回 True，否则返回 False。注意，如果用户不在 M 函数文件中使用任何锁定命令，则该函数的默认设置是没有锁定的。

（4）用户可以使用 pcode 命令将编译后的伪码存储到硬盘上。该命令执行后，Matlab 将会把编译后的函数，而不是 M 文件，载入到内存。对于大多数的函数来说，伪码文件在第一次运行时，并不能显著地缩短函数的执行时间，但是，当用户反复调用该函数，或者当该函数需要与复杂的图形用户界面进行交互时，采用伪码文件的确可以大大提高函数的运行速度。用户可以使用下面的命令创建伪码文件：

```
>> pcode myfunction
```

其中，myfunction 是需要编译的 M 文件的名字。所谓伪码文件是一个经过加密的、与操作系统无关的二进制文件，该文件原始 M 文件具有相同的文件名，但其后缀是.p，而不是.m。利用伪码文件运行函数可以确保函数源码本身的安全性，因为所有的伪码文件都是由一些特殊的二进制代码构成的，一般的浏览器是无法解释这些代码的。另外，伪码文件是无法被转换为 M 文件的。由于伪码文件所具有的二进制特性，伪码文件在 Matlab 各版本中是无法后向兼容的。也就是说，在 Matlab 7.0 中创建的伪码文件是无法在 Matlab 6.X 中使用的，但在 Matlab 6.X 中创建的伪码文件可以在 Matlab 7.0 中使用。

（5）本书第 3 章讲到，当 Matlab 遇到了一个它不认识的名称时，它将遵循一套优先级规则来寻找该名称所代表的对象，然后执行相应的操作。例如，当 Matlab 遇到一个名为 cow（该名称可能是用户在命令窗口中输入，或是包含在一个 M 脚本文件或 M 函数文件中）的指令时，便按照下面的优先级顺序寻找 cow 所代表的对象：

a. 首先检查 cow 是不是当前工作区中的一个变量，如果不是，执行下一步。
b. 再检查 cow 是不是 Matlab 的内置函数，如果不是，执行下一步。
c. 再检查 cow 是不是其所在文件中的一个子函数，如果不是，执行下一步。
d. 再检查 cow 是不是其所在文件中的一个私有函数，如果不是，执行下一步。
e. 再检查 cow 是否存在于当前目录下，如果不是，执行下一步。
f. 最后按照路径列表的顺序检查 cow 是否存在于 Matlab 的其他搜索路径中的目录或子目录中。
g. 如果以上均不成立，则 Matlab 返回一个错误。

注意，在步骤 e、f 中，Matlab 对不同的文件类型还有优先级划分，其中 MEX 文件优先级最高，伪码文件次之，最低的是 M 文件。例如，如果在步骤 d 中，cow.mex、cow.p 和 cow.m 都存在，那么 Matlab 就会使用 cow.mex，如果只存在 cow.p 和 cow.m，那么 Matlab 将会使用 cow.p，如果只有 cow.m 存在，并且 cow.m 又不是 Matlab 的内置函数，则 Matlab 才会使用该文件。

（6）当 Matlab 启动时，将把存储在 toolbox 目录及其子目录中的所有 M 文件的名字和位置存入高速缓冲区中，以提高函数的运行速度。toolbox 目录及其子目录中的文件被存入缓冲区后，这些文件都被视为只读属性。也就是说，当一个函数被执行后，再对其进行修改，则 Matlab 只执行原来的函数，忽略对这个函数所做的改变。另外，如果一个新的 M 文件在 Matlab 启动之后才被加入到 toolbox 目录或其子目录下，那么它们并不会被载入到高速缓冲区中。

因此，在 M 函数文件开发完成之前，最好把它们保存在 toolbox 以外的目录中，例如 Matlab 安装目录；当这些函数开发完成后，再把它们再存储到只读的 toolbox 目录或其子目录中；最后，修改 Matlab 的搜索路径，以便使 Matlab 下次启动时能够找到这些 M 函数文件，并把它们载入到高速缓冲区中。

（7）一个新的 M 函数文件被载入到高速缓冲区中以后，并不能立即被调用，只有使用 rehash toolbox 命令对高速缓冲区进行刷新后，Matlab 才能找到并调用它。另一方面，当一个已被载入高速缓冲区中的文件被修改时，Matlab 也不能立即识别这些修改，只有通过命令 clear 将原来的函数从内存中清除之后，Matlab 才能知道这些变化。要将一个未锁定的文件从内存中清除，可以使用如下的命令：>>clear myfun，用户也可以使用命令>>clear functions 将所有未锁定的函数从内存中清除出去。注意：clear 命令只能清除未锁定的函数文件，不能清除已锁定的函数文件。

（8）对于 toolbox 目录之外的 M 文件，Matlab 通过辨别其修改日期判断执行新函数还是原来的函数。如果一个 M 文件函数已经被编译到内存，当 Matlab 再次调用该函数时，将会比较此时硬盘上的 M 函数文件的修改日期与内存中 M 函数文件的修改日期，如果这两个日期是一样的，Matlab 将会执行内存中的 M 文件，如果硬盘上的 M 函数文件比内存中的 M 函数文件要新，Matlab 就会将原来的编译文件清除，然后编译新的经过修改后的文件并执行。

（9）利用函数 depfun，可以检查一个 M 函数文件与其他所有文件之间的文件相关性。该函数严格列出了一个 M 文件对其他 M 文件函数、内置函数的调用以及 eval 字符串中的函数调用和回调，还指出了函数所用到的变量和 Java 类。当安装了不同 Toolboxes 的用户之间共享 M 函数文件时，用该函数标识出一个函数与另一用户的所有函数之间的相关性是非常有用的。另外，函数 depdir 用来返回一个 M 函数文件的相关目录的列表，该函数内部用到了 depfun 函数。

12.5　创建用户自己的工具箱

有 Matlab 编程经验的用户都知道，当用户利用 Matlab 完成一项任务时，通常都需要将用户创建的与该任务相关的 M 文件存放在一个目录中，并将该目录加入到 Matlab 的搜索路径中，或者将这些文件直接存放在 Matlab 搜索路径中的一个子目录中，但很少有用户想到将这些 M 文件存放在 toolbox 目录中的一个子目录下。实际上，如果用户已经将这些 M 文件编辑完成，就可以在 toolbox 目录中创建一个子目录作为用户自己的工具箱，然后将这些 M 文件放在该子目录中，当 Matlab 再次启动时，这些文件便被载入到高速缓冲区中，这样就可以大大提高这些文件的执行速度。为了便于文件管理和日后的维护，当用户

创建自己的工具箱时，最好添加两个 M 脚本文件：Readme.m 和 Contents.m，这两个文件并不包含任何操作命令，而全部由一些注释行构成。下面对这两个文件进行简单描述：

（1）Readme.m 文件通常应包含对工具箱所作更新的描述以及对某些函数特性进行的补充说明，当然也可以包含其他内容，这将根据用户的需要而定。在命令窗口输入命令>>whatsnew MyToolbox（其中 MyToolbox 是用户创建的工具箱名称，即包含 M 文件的子目录名称），就会在 Help 窗口中显示 Readme.m 文件的具体内容。如果用户想把这个 Toolbox 上载到 Mathworks 的网站上，Readme.m 文件中还应该包含一个如下的声明，以避免法律上的纠纷：

```
% These M-files are User-Contributed Routines that are being
% redistributed by The Mathworks, upon request, on an "as is " basis. A
% User-Contributed Routine is not a product of The Mathworks, Inc. and
% The Mathworks assumes no responsibility for any errors that may exist
% in these routines.
```

（2）Contents.m 文件主要包含 Toolbox 目录中所有 M 文件的名称。在命令窗口中输入命令>>helpwin MyToolbox（其中 MyToolbox 是用户创建的工具箱），就会在 Help 窗口中显示出工具箱中的文件列表。Contents.m 文件的第一行通常声明了工具箱的名字，第二行则声明了该工具箱的版本及日期，如下所示：

```
% Toolbox Description
% Version xxx dd-mmm-yyyy
```

使用 ver 命令就可以查看这些信息，ver 命令将返回所有已安装的工具箱的名称、版本信息及时间列表。

（3）一般情况下，用户总是经常使用自己编写的工具箱或者某几类工具箱，而在同一工具箱中，又总是经常使用某些函数，因此，总希望这些工具箱或函数在使用时具有比其他工具箱或函数更高的优先权。Matlab 最早的一种解决方法是将各工具箱或函数的优先权信息保存到一个 MAT 文件中，然后在 Matlab 的执行周期中调用此文件，从而获取优先权信息，这种方法需要为 MAT 文件选择一个合适的目录（该目录应在 Matlab 的搜索路径中），并且要保证其不被删除。为了消除 MAT 文件方法的脆弱性，Matlab 专门提供了函数 getpref、setpref、addpref 和 rmpref 分别获取、设置、添加和删除工具箱或函数的优先权。Matlab 将优先权相同的工具箱或函数分成一组，并给该组取一个优先权名称，名称的值可以是任何的 Matlab 变量。Matlab 保存优先权信息的方式有些类似于结构体的工作方式。例如，命令 group.prefname=values 指将 group 组的优先权命名为 values。用户在通过这些文件处理优先权时，并不直接对优先权文件（MAT 文件）进行操作，优先权文件的存储位置完全由系统决定，这样就保证了该文件的安全性，使它们在 Matlab 的各个执行周期始终存在。

12.6 命令-函数的二元性

用户除了可以创建 M 函数文件外，还可以创建 Matlab 命令。Matlab 命令读者应该也不陌生，clear、who、dir、ver、help 和 whatsnew 都是 Matlab 命令。Matlab 命令与 M 函数非常类似，只有两个地方稍有差别：

（1）Matlab 命令没有输出参数。

（2）Matlab 命令的输入参数没有包含在括号内，而是直接跟在命令名后面。例如，clear functions 就是一条命令，它接受不带括号的输入参数 functions，其功能是将所有经过编译的函数从内存中清除，该命令没有输出。而一个函数则与之不同，它通常将计算结果赋给一个或者多个输出参数，并且必须用逗号将其输入参数隔开，然后用括号括起来，例如，a=atan2(x,y)就是一个函数。

实际上，Matlab 命令可以看作是满足上述两点差别的特殊的“函数调用”。例如，命令 whatsnew 实际上就是一个 M 函数文件，当它在命令窗口作如下调用时：

```
>> whatsnew MyToolbox
```

Matlab 就把该命令解释成下述 whatsnew 函数的调用：

```
>> whatsnew ('MyToolbox')
```

换言之，只要不要求有输出参数，Matlab 就将命令参数作为一个字符串参数放进括号里，然后将命令视为一个函数进行调用，并且这种解释方法适用于所有的 Matlab 命令。

一个命令既可以以命令的形式在命令窗口中输入，也可以以函数的形式在命令窗口中输入。例如，对于 which 命令，当以命令形式输入时，格式如下：

```
>> which fname
```

该命令将显示 fname 文件所在路径的一个字符串。当以函数形式输入时，格式如下：

```
>> s = which('fname')
```

该语句将 fname 所在路径的字符串赋给变量 s。但下面的语句将是行不通的：

```
>> s = which fname
??? s = which fname
              |
Error: Missing MATLAB operator
```

上面的语句之所以会出现错误，是因为命令是不能有输出参数的。在任何情况下，Matlab 如果遇到了一个等号，就把后面的语句解释成一个函数，而函数的调用则必须将输入参数包含在一个括号内，并以逗号隔开。

总的来说，命令和函数都可以以函数调用的形式存在。当将命令写成函数调用形式时，首先把命令参数解释成字符串并放到括号中，然后再调用同名函数。另一方面，如果一个函数不要求产生输出，并且只要求一个字符串输入，则该函数也可以以命令的方式来调用。

12.7　函数句柄和匿名函数

在很多情况下，用户需要将一个函数的标识作为参数传递给另一个函数。比如，Matlab 的许多数值分析函数就需要将用户提供的一个函数作为函数输入参数进行验证。例如，本书第 24 章我们将讲到 quad 函数，它的一个调用方式为：quad(Fun,low,high)，其功能是利用 Fun 函数计算[low high]范围内的面积值。在 Matlab 的早期版本中，Fun 都是由表征一个函数的字符串表示的，比如，sin(x)通常由'sin'表示。从 Matlab 5 开始引入了内联函数（inline

function），它从一个字符串表达式中创建函数。例如，下面的代码创建了一个内联函数 il_humps：

```
>> il_humps = inline('1./((x-.3).^2+.01)+1./((x-.9).^2+.04)-6','x')
il_ humps =
    inline function:
    il_humps(x) = 1./((x-.3).^2+.01)+1./((x-.9).^2+.04)-6
```

上例中，函数 inline 从一个字符串创建一个函数，并以 x 为输入变量。要在一个函数中调用内联函数，只需将该内联函数的名字作为输入参数传递给函数即可。例如，要想将 quad(Fun,low,high)中的 Fun 换为上面的内联函数 il_ humps，只需按下面的方式调用即可：quad(il_ humps,low,high)。

要验证一个由字符串表示的函数或一个内联函数，可以使用 feval 函数。下面的代码分别验证了正弦函数和前面创建的 il_ humps 函数：

```
>> y = feval('sin',pi*(0:4)/4)
y =
          0      0.70711      1        0.70711      1.2246e-016
>> z = feval(il_humps,[-1 0 1])
z =
    -5.1378      5.1765      16
```

feval 函数只将第一个输入参数视为要验证的函数，而将其他的参数视为附加参数。本节后面还将给出 feval 更详细的例子，实际上，feval 函数可以接受任意个数的输入和输出参数。

虽然 Matlab 7 仍然支持用字符串代表函数，也支持内联函数，但由于该版本引入了一个新的函数类型：匿名函数（anonymous function），并用函数句柄（function handle）来代表它，因此，现在并不鼓励用户使用前面两种方法，而尽量使用匿名函数句柄来引用函数。

下面的代码给出了一个匿名函数的例子：

```
>> af_humps =@(x) 1./((x-.3).^2+.01)+1./((x-.9).^2+.04)-6;
```

其中，@符号意味着等号左边是一个函数句柄。@后面的(x)定义了函数的输入参数，最后一部分是函数表达式。我们同样可以使用 feval 函数来验证匿名函数，例如，可以使用下面的代码验证 af_humps：

```
>> z = feval(af_humps,[-1 0 1])
z =
     -5.1378     5.1765      16
```

其实，用户根本没有必要利用 feval 函数来验证匿名函数，因为匿名函数可以使用自己的函数句柄直接进行验证，例如，上面的例子可以简写为：

```
>> z = af_humps([-1 0 1])
ans =
     -5.1378     5.1765      16
```

匿名函数在定义过程中可以调用任何 Matlab 函数（包括用户自定义的函数），也可以使用当时 Matlab 工作区中存在的任何变量。例如，下例中的匿名函数 af_humpsab 在定义

时就使用了 Matlab 工作区中的变量 a 和 b：

```
>> a = -.3; b = -.9;
>> af_humpsab = @(x) 1./((x+a).^2+.01)+1./((x+b).^2+.04)-6;
>> af_humpsab([-1 0 1])
ans =
     -5.1378     5.1765     16
```

我们看到，af_humpsab 在定义时引用了前面定义的变量 a 和 b。应当注意，当 a 或 b 的值变化时，匿名函数并不改变，这是因为函数句柄只捕捉它创建那一时刻的变量的值，并不随变量的变化而变化。例如，如果将 a 的值改为 0，再重新验证 af_humpsab，会得到下面的结果：

```
>> a = 0; % changing the value of a does not change the function
>> af_dumpsab([-1 0 1]) % evaluate again, get the same results
ans =
     -5.1378     5.1765     16
```

我们也可以针对一个内置函数或一个 M 文件函数来创建匿名函数句柄。下面的函数分别基于 M 文件函数 humps 和 Matlab 内置函数 cos 创建了两个文件句柄：

```
>> fh_Mfile = @humps   % function handle for M-File function
fh_Mfile =
    @humps

>> fh_Mflie(1)  % evaluate humps(1)
ans =
    16
>> fh_builtin = @cos  % function handle for built-in function
fh_builtin =
    @cos

>> fh_builtin(pi)  % evaluate cos(pi)
ans =
    -1
```

有上例可知，要创建一个内置函数或一个 M 文件函数的句柄也很容易，只要在等号右边使用@符号，并在该符号后紧跟内置函数名或 M 文件函数名即可。

我们还可以利用单元数组同时创建多个内置函数和 M 文件函数的句柄，验证这些函数时，只需引用该函数所在单元即可。例如，下面的代码将上面两个独立创建的句柄利用一个单元数组 fhan 创建在了一起：

```
>> fhan = {@humps @cos}
fhan =
    [@humps]  [@cos]

>> fhan{1}(1)   % evaluate humps(1)
ans =
    16
>> fhan{2}(pi)  % evaluate cos(pi)
ans =
    -1
```

上例中，fhan 包含了 M 文件函数 humps 和内置函数 cos 的文件句柄，并将它们分别存放到不同的单元中。验证这些函数时，只要对这些单元进行索引即可。在 Matlab 6 中，函数句柄是采用标准数组的形式创建的，例如上例第一条语句在 Matlab 6 中将写为：fhan=[@humps @cos]。目前，在 Matlab 7 中已不再支持这种标准数组的创建方法。

Matlab 还专门提供了一些函数来处理和应用句柄。例如，函数 functions 将返回一个句柄的详细信息，如下例所示：

```
>> functions(fh_Mfile)
ans =
    function: 'humps'
        type: 'simple'
        file: 'C:\matlab7\toolbox\matlab\demos\humps.m'
>> functions(fh_builtin)
ans =
    function: 'cos'
        type: 'simple'
        file: 'C:\matlab7\toolbox\matlab\elfun\cos.m'
>> functions(af_humps)
ans =
    function: '@(x)1./((x-.3).^2+.01)+1./((x-.9).^2+.04)-6'
        type: 'anonymous'
        file: ''
   workspace: [1×1 struct]
```

需要注意的是，functions 函数通常只在调试程序时使用，因为它的返回值很容易发生变化。

当我们将一个函数名作为字符串传递给函数 str2func 时，也可以创建该函数的函数句柄。我们看下面的例子：

```
>> myfunc = 'humps' % place name of humps.m in a string variable
myfunc =
humps
>> fh2 = @myfunc  % This doesn't work!
??? Error: "myfunc" was previously used as a variable,
conflicting with its use here as the name of a function.
>> fh2 = str2func(myfunc) % this works
fh2 =
    @humps
>> isequal(fh2,fh_Mfile) % these are equal function handles
ans =
    1
```

func2str 函数将执行与 str2func 函数相反的操作，它从一个函数句柄中提取函数名，并保存到一个字符串变量中。当函数句柄是一个内置函数或 M 文件函数句柄时，该函数返回函数的名称，当函数句柄是一个匿名函数句柄时，该函数返回匿名函数的函数表达式。下例利用 func2str 函数分别从句柄 fh2 和 af_humps 中获取函数的名称：

```
>> func2str(fh2) % M-file function
ans =
```

```
humps
>> func2str(af_humps) % anonymous function
ans =
@(x)1./((x-.3).^2+.01)+1./((x-.9).^2+.04)-6
>> class(ans) % output is character string
ans =
char
>> isa(fh2,'function_handle') % True for function handles
ans =
    1
```

函数句柄是 Matlab 提供给用户的一个强有力的工具。首先，当一个函数句柄被创建时，它将记录函数的详细信息（见前面 functions 函数返回的结果），因此，当使用一个函数的句柄调用该函数时，Matlab 会立即执行，不再需要进行文件搜索。尤其是当反复调用一个文件时，可以节省大量的搜索时间，从而提高函数的执行效率。

函数句柄的另一个重要特性是它们可以用来标识子函数、私有函数和嵌套函数。一般情况下，这些函数对于用户来说都是“隐蔽”的，这些标识对于用户正确使用这些函数非常有用。例如，当我们在编写一个含有子函数的 M 文件函数时，可以为子函数创建一个句柄，并作为主函数的一个输出参数提供给用户，这样就使本来“隐蔽”的子函数“显现”出来，便于用户对其进行验证和使用。下面的 M 文件函数框架（因为该函数并不执行实际有意义的操作）向读者提供了一个在主函数中返回子函数句柄的例子：

```
function out=myfunction(select)
%MYFUNCTION Return function handle to a subfunction.
% Example function demonstrating function handles to subfunctions.

switch select
case 'case1'
   out = @local_subfun1;
case 'case2'
   out = @local_subfun2;
otherwise
   out = [];
   error('Unknown Input.')
end
function a=local_subfun1(b,c)
%LOCAL_SUBFUC Some function operation.
% code that operate on the input arguments b and c
% and returns content in the variable a

% end of local_subfun1
function d=local_subfun1(e,f)
%LOCAL_SUBFUC Some function operation.
% code that operates on the input arguments e and f
% and returns content in the variable d

% end of local_subfun2
```

下面的代码通过设置 select 参数的值获取一个子函数的句柄，并在主函数的调用环境

中调用这个子函数：

```
>> h_subfun = myfunction('case2'); % handle to local_subfun2
>> dout = h_subfun(x,y); % execute local_subfun2(e,f)
```

上述代码之所以能运行，是因为子函数句柄记录了子函数的所有信息，包括子函数的声明、代码和位置等。

12.8 嵌套函数

嵌套函数是 Matlab 7 新增的一种函数类型。这种函数对于熟悉它的编程人员是十分有用的，但如果用户对该函数类型不是十分了解，并对其使用不当，反而会造成程序质量的下降。因此，本节有必要通过一些简单的讲解和示例使读者对嵌套函数有个整体认识。

嵌套函数不需要使用全局变量和输入输出参数就可以在函数之间传递数据信息。下面的 M 函数文件给出了一个嵌套函数的简单例子：

```
function out = primary_function(...)
%PRIMARYFUNCTION primary function.

% code in primary function.
% this code can call nested functions

   function nout1=nested_function1(...)
   % code in nested_function1.
   % In addition to variables passed through the input arguments
   % this nested function has access to all variables in existence
   % in the primary function at the point where this function
   % definition appears. This access permits reading and writing
   % to all variables available.

   end  % required to mark the end of nested_function1
% other code in primary_function, including other
% nested functions terminated with end statements

end % end statement required here for primary_function
```

从 primary_function 函数文件可以看到，嵌套函数是完全置身于另一个函数定义中的函数。如果一个函数包含嵌套函数，则无论嵌套函数还是主函数都必须以 end 作为函数的结束。嵌套函数与子函数都是出现在一个主函数的定义之中的函数，但它们之间有明显的区别，主要表现在：①嵌套函数可以出现在主函数声明之后的任何位置，而自函数只能出现在主函数体之后。②嵌套函数可以访问主函数工作区中的任何变量值，而子函数只能通过输入参数访问主函数工作区中的某些变量。③主函数可以访问嵌套函数工作区中的所有变量，而只能通过子函数的输出参数访问子函数工作区中的某些变量。

原则上，一个函数可以包含任意多的嵌套函数，并且嵌套函数还可以包含嵌套函数。一个嵌套函数既可以通过输入参数访问主函数工作区中的变量，还可以直接访问主函数工作区中的其他变量。这些内容也许会令人费解，实际上，嵌套函数使得 M 文件的调试变得非常复杂，并且初学者基本上无法感觉到它的优势究竟在什么地方。只有等到读者上升到

一个 Matlab 高手时，也许会对 Matlab 这一新功能有一定的认识。

下面给出了一个嵌套函数的典型应用。在该函数中，嵌套函数直接从主函数的工作区中获取变量值，计算一个有理多项式的值，而主函数则返回嵌套函数的函数句柄。

```
function fhandle=nestexample(num,den)
%NESTEXAMPLE Example Nested Function.
% NESTEXAMPLE(Num,Den) returns a function handle to a function that
% can be used to evaluate a rational polynomial. Num and Den are vectors
% containing the numerator and denominator polynomial coefficients.
%
% For example, ratpoly=nestexample([1 2],[1 2 3]) returns a function
% handle that facilitates evaluation of the rational polynomial
%
%     x + 2
%   -----------
%   x^2 + 2x +3
if ~isnumeric(num) || ~isnumeric(den)
    error('Num and Den Must be Numeric Vectors.')
end
num=reshape(num,1,[]); % make num into a row vector
den=reshape(den,1,[]); % make den into a row vector
fhandle=@nested_ratpoly; % create function handle for return
    function out=nested_ratpoly(x)
    % Nested function that evaluates a rational polynomial, where the
    % numerator and denominator coefficients are obtained from the
    % primary function workspace. Only the evaluation points x appears
    % as an input argument.

    out = polyval(num,x)./polyval(den,x);

end % nested function terminated with an end statement
end % primary function terminated with an end statement too!
```

使用上面的函数 nestexample，我们可以通过下面的代码创建嵌套函数的句柄，从而计算有理多项式的值，并利用 plot 函数画出相应的曲线：

```
>> ratpoly1 = nestexample([1 2],[1 2 3]) % (x+2)/(x^2+2x+3);
>> ratpoly2 = nestexample([2 1],[3 2 1]) % (2x+1)/(3x^2+2x+1);
>> x = linspace(-10,10);    % independent variable data
>> y1 = ratpoly1(x);        % evaluate first rational polynomial
>> y2 = ratpoly2(x);        % evaluate second rational polynomial
>> plot(x,y1,x,y2)          % plot created data
>> xlabel('X');
>> ylabel('Y');
>> title('Figure 12.1: Rational Polynomial Evaluation')
```

上例两次调用 nestexample 函数，分别创建了计算有理多项式 $\frac{x+2}{x^2+2x+3}$ 和 $\frac{2x+1}{3x^2+2x+1}$ 的嵌套函数的句柄。由于嵌套函数中的多项式已通过函数句柄从 nestexample 的输入参数中获取系数，因此，在利用句柄调用它们时，就不用再提供多项式系数参数。

上例绘制的图形如图 12.1 所示。

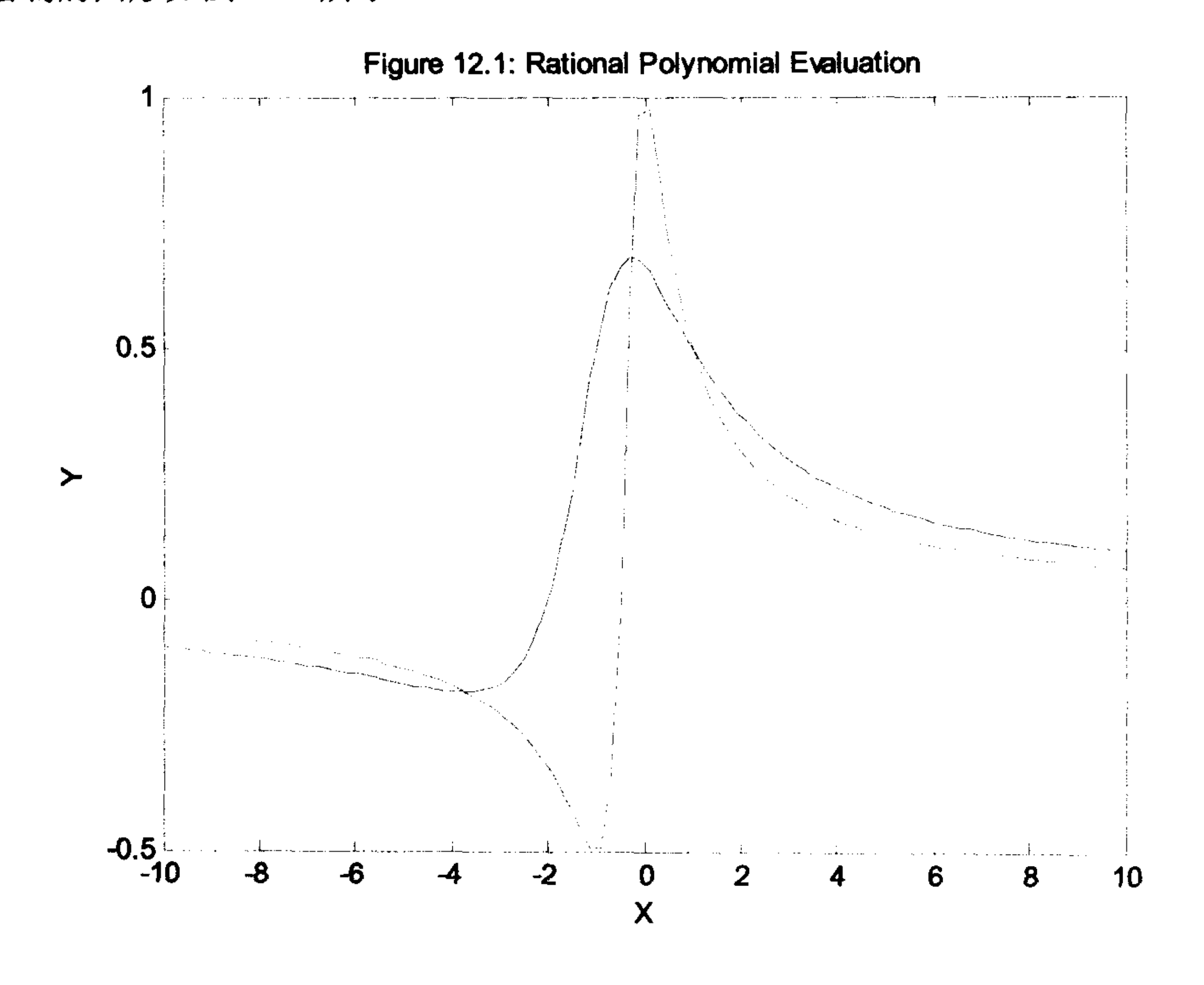


图 12.1 有理多项式计算

我们也可以不使用函数句柄，而利用函数的输出值计算多项式的值，下面的 M 文件函数便是一个例子：

```
function out=ratpolyval(num,den,x)
%RATPOLYVAL Evaluate Rational Polynomial.
% RATPOLYVAL(Num,Den,X) evaluates the rational polynomial, whose
% numerator and denominator coefficients are given by the vectors
% Num and Den respectively, at the points given in X.
if ~isnumeric(num) || ~isnumeric(den)
   error('Num and Den Must be Numeric Vectors.')
end
num=reshape(num,1,[]); % make num into a row vector
den=reshape(den,1,[]); % make den into a row vector
out=polyval(num,x)./polyval(den,x);
```

利用 ratpolyval 函数，我们可以采用下面的代码计算前面两个多项式的值：

```
>> yy1 = ratpolyval([1 2],[1 2 3],x); % same as y1 above
>> yy2 = ratpolyval([2 1],[3 2 1],x); % same as y2 above
```

可以看到，上述调用方式每次都需要给函数 ratpolyval 提供多项式的系数。这种方式虽然看起来比较直观，也容易理解，但不利于调用，尤其是当用户需要反复调用时，每次都要输入多项式的系数显然是一件非常繁琐的事。嵌套函数则利用句柄将多项式的信息封装

在一个供 Matlab 内部使用的单元数组中，因此，用户只需在第一次创建函数句柄时提供多项式系数，以后每次调用时都不需要再输入这些系数，使函数调用变得简捷方便。

由于有理多项式可以用一个简单的表达式表示，因此，上面的例子也可以使用匿名函数进行计算。例如，下面的代码利用匿名函数句柄创建前面的两个多项式：

```
>> num = [1 2]; den = [1 2 3]; % specify numerator and denominator
>> ratpoly1 = @(x) polyval(num,x)./polyval(den,x); % create anonymous fun

>> num = [2 1]; den = [3 2 1]; % redefine numerator and denominator
>> ratpoly2 = @(x) polyval(num,x)./polyval(den,x); % create anonymous fun

>> y1 = ratpoly1(x); % evaluate first rational polynomial
>> y2 = ratpoly2(x); % evaluate second rational polynomial
```

通过上面的比较可以看出，利用嵌套函数句柄来验证一个函数是非常方便的。当用户需要对一个函数进行反复验证时，利用该方法可以创建该函数的多个实例（即对函数的多次调用），并且每个实例都有自己的工作区，各实例之间互不干扰。

Chapter 13

M 文件的调试和剖析

用户在开发 M 函数文件的过程中，不可避免地会出现错误，也就是所谓的 bug。为此，Matlab 提供了一些方法和函数用于 M 文件调试。另外，Matlab 还提供了一个有效工具——剖析器——帮助用户提高 M 文件的执行速度。M 文件剖析器通过标识哪一行代码的运行时间最长，来优化 M 文件的执行，提高其执行速度。

13.1 调试工具

在 Matlab 中，通常存在两种类型的错误：语法错误和运行错误。语法错误一般是由于用户的错误操作造成的，例如变量或函数名拼写错误、缺少引号或括号等，当 Matlab 运行一个表达式或一个函数被编译进内存时，都会发现这些错误。Matlab 发现错误后，便立即标识出这些错误，并向用户提供错误的类型以及错误在 M 文件中的位置（行的标号）。利用这些信息，用户能很方便地对错误进行定位，并纠正它。有一种语法错误例外，它出现在 GUI 回调字符串中。如果用户将 GUI 回调字符串拼错，在 GUI 运行过程中，如果该字符串本身不被执行，Matlab 就无法发现这些错误。

运行错误能被 Matlab 发现并标记出来，但用户很难发现这些错误到底发生在何处。当 Matlab 发现运行错误后，便立刻返回到命令窗口和 Matlab 工作区。这时，用户无法访问错误发生时的函数工作区内的值，也就不能通过查询函数工作区的方法筛查错误来源。因此，检查运行错误的一个最好的办法是依靠编程经验和他人的指导。

根据作者的经验，最常见的运行错误大都发生在某项运算得出了空数组或者 NaN 结果时。第 10 章讲到，针对 NaN 的任何运算都会返回 NaN，因此，如果用户怀疑是由于 NaN 引起的运行错误，最好的解决办法是利用逻辑函数 isnan 来检测是否在运算过程中出现了 NaN 结果。第 10 章还讲到，由于空数组的维数为 0，因此对空数组的寻址必然导致错误。通常会产生空数组结果的是 find 函数，因为当 find 无法找到指定的数据时，就会返回空数组。空数组具有“扩散性”，也就是说，如果对一个空数组进行数学运算，结果将仍是空数组。例如，下面的代码演示了空数组的“扩散性”：

```
>> x = pi*(1:4)             % example data
x =
```

```
    3.1416      6.2832      9.4248      12.5664
>> i = find(x>20)           % use find function
i =
    Empty matrix: 1-by-0
>> y = 2*x(i)               % propagate the empty matrix
y =
    Empty matrix: 1-by-0
```

显然，当用户希望 y 是有限维数组时，就会出现一个运行错误。当我们使用有可能产生空数组结果的函数进行运算时，利用逻辑函数 isempty 来检测计算中是否出现空数组结果是很有必要的，因为这样可以最大限度地避免出现运行错误。

M 函数文件的调试方法很多。对于简单的问题，我们可以直接使用下述的一种或几种方法来调试：

（1）将函数中要调试的语句行后面的分号去掉，这样运算的中间结果就可以在命令窗口中显示。

（2）在函数中添加额外的语句，用于显示要查看的变量。

（3）在 M 函数文件中选定的位置添加 keyboard 命令，当函数执行到 keyboard 命令时，便停止执行，并将临时控制权交给键盘。这时，用户就可以查看函数工作区中的变量，并可以根据需要改变变量的值。要恢复函数的执行，只要在键盘提示符（K>>）后输入 return 即可。

（4）在 M 函数文件的函数声明语句（即第一行语句）之前插入%，可以将第一行变为注释行，同时也把 M 函数文件变为 M 脚本文件。这样，在文件执行时，其工作区就是 Matlab 工作区，用户在出现错误时就可以查看工作区中的变量。

当一个 M 函数文件容量很大、递归调用或者是高度嵌套（也就是说该函数调用其他的 M 文件函数，而这些函数又调用其他的函数，依此类推）时，用户可以使用 Matlab 提供的图形化调试函数，这些调试函数存在于 Editor/Debugger 窗口中的 Debug 和 Breakpoints 菜单上。当然，用户也可以在命令窗口中直接输入与这些调试函数等价的函数或命令，只不过相对比较麻烦。用户可以通过输入下面的命令查看这些等价函数的联机帮助：

```
>> doc debug
```

用户可以通过图形化调试工具设置函数的运行和停止，例如，用户可以使函数在设定的断点处停止运行，或者使函数在出现告警和错误的地方停止运行，以及在得到 NaN 和 Inf 结果的地方停止运行，等等。如果用户在程序中设置了断点，则当程序运行到断点时，Matlab 在执行完断点所在的行之后立即停止执行，并在 Matlab 工作区中显示当前各内部变量的值。程序停止运行后，命令窗口会出现键盘提示符 K>>，这时用户可以查询函数工作区，改变工作区中的变量的值，还可以根据需要进行跟踪调试。另外，Editor/Debugger 窗口还将显示目前函数执行到哪一行，并提供了用户访问被当前调试的 M 文件函数调用的其他函数的工作区的方法。当用户在函数中设置好了断点，Editor/Debugger 窗口提供了一系列菜单让用户利用该断点进行调试，这些菜单包括：单步执行菜单、执行到下一个断点菜单、执行到光标位置菜单、结束调试菜单。

根据调试的文件不同，加上不同的用户在调试程序时有不同的习惯，因此，很难用一两句话来描述图形化调试工具的用法。不过，Matlab 所提供的这些工具是非常直观且简单易用的，用户很容易掌握其应用要领，充分享受 Matlab 图形化调试工具在创建高质量代码过程中的便捷和高效。

13.2 语法检查和文件相关性

当用户创建和调试 M 文件时，适时地检查代码中的语法错误和运行错误是非常必要的。过去，我们通常需要执行完 M 文件，然后再从命令窗口中返回的告警和错误信息判断是否存在语法错误或运行错误，或者使用 pcode 命令创建一个 P-码文件来寻找错误来源。在 Matlab 7 中，用户可以使用新增加的 mlint 函数来分析 M 文件中的语法错误以及其他可能存在的问题或不完善的地方。例如，mlint 可以检查出文件中是否有变量虽然被定义但却从未调用，是否有输入参数没有使用，是否有输出参数没有赋值，是否存在已不再使用的函数或命令，是否存在 Matlab 根本执行不到的语句，等等。该函数通常都是以命令的形式调用的，其调用格式为：

```
>> mlint myfunction
```

其中，myfunction 是要检查的 M 文件的名称。当然，该函数也可以以函数的形式进行调用，用户可以通过多种方式获得其输出结果。例如，下面的代码将 mlint 的输出结果赋给了一个结构体变量 out：

```
>> out = mlint('myfunction','-struct');
```

除了可以利用 mlint 检查一个文件外，用户还可以利用 mlintrpt 函数检查某一目录下的所有文件，并生成一个新的 HTML 类型的窗口显示每个文件的检查结果。另外，在 Matlab 桌面中的当前路径窗口中，用户也可以使用这一特性。

由于现在 M 文件很容易通过存储设备（如优盘、软盘等）或网络由一台计算机传递到另一台计算机，这样有时就会出现这种现象：当一个 M 文件在执行时，由于找不到一个或多个被调用的 M 文件函数而导致这个 M 文件运行失败。这很可能是由于用户只传递了主函数，而没有将主函数调用的其他相关函数传递过来，结果就导致了 Matlab 在指定的搜索路径中找不到这些被调用的函数，从而使主函数运行失败。为了发现这一问题，Matlab 提供了函数 depfun 用于解析 M 文件的相关性。该函数递归地搜索所有的文件关联性，包括被 M 文件调用的函数的关联性，以及该文件通过图形句柄对象被别的文件调用的关联性。由于该函数将产生大量的输出信息，因此这里就不再对其进行举例，读者最好亲自在 Matlab 中验证一下。

13.3 M 文件剖析

即便是在 M 函数文件正常工作的情况下，也需要用一些方法来对代码进行优化，以避免不必要的计算或者函数调用。例如，将一次计算的结果保存起来，就可以避免复杂的重复计算；使用向量技术就可以避免诸如 For 循环那样的迭代过程，从而提高代码性能；使

用 Matlab JIT 加速器就可以缩短程序的运行时间；等等。大多数用户在编写一个程序时，自己都不知道到底程序的运行时间是怎么分配的。随着当今高速浮点处理器的出现，直接计算一个结果也许要比先将数据保存在一个变量中然后再调用这些数据进行计算要快得多。因此，在内存使用和计算次数之间往往存在一个内在的折衷。根据处理的数据不同，有时用更多的内存来存储中间结果会更快一些，有时执行更多的运算会更快一些。但无论如何，这个折衷往往都依赖当前处理的数据集的大小，处理大量数据的最优方法和处理少量数据有很大的不同。而且我们通常会发现，编程时的优化算法往往与书面上的理论优化算法大相径庭，这往往又是问题实现的一大障碍。

为了使用户编写的程序尽可能高效实用，Matlab 提供了一系列剖析工具来优化 M 文件函数的执行。当一个 M 文件开始执行时，这些工具就开始监视该文件的执行过程，并记录哪些语句行占用的时间最多。例如，如果某行语句（或某个函数调用）占用了 M 文件 50% 的执行时间，那么对这行语句（或函数调用）的优化程度将会对整体 M 文件的执行速度产生十分显著的影响。这时，用户就需要格外注意对这行代码的优化，例如，用户可以重新编写该行代码以消除或替换可能影响运行速度的代码；也可以将该行要处理的数据量减少到最小，以提高运行速度；或许用户根本不知道如何再对其进行优化，但也要保证这是你所能做的最好的优化。因此，无论在什么情况下，剖析 M 文件函数的运行都可以使用户获得对该函数更深入的认识，使函数尽可能达到较高水平。

除了可以使用 Matlab 桌面上 Profiler 窗口中的剖析器之外，用户还可以使用 profile 命令来确定一个 M 文件中哪一行占用了最多的执行时间。例如，我们可以通过下面的命令剖析函数 myfunction 的执行时间：

```
>> profile on

>> for i = 1:100
       out = myfunction(in);
   end

>> profile viewer
```

上述命令首先打开剖析器（profiler），然后利用一个 for 循环使 myfunction 函数执行足够多的次数（100 次），最后生成了一个剖析结果。该剖析结果是一个在 Profiler 窗口中显示的 HTML 文件，它包含各种各样的数据，点击文件中不同的链接就可以查看不同的剖析信息。

Chapter 14

文件和目录管理

Matlab 7 可以打开和保存多种格式的文件，有的文件格式是为 Matlab 定制的，有的则是业界通用的标准文件格式，另外还有一些是为其他应用程序定制的文件格式。在 Matlab 中，用户可以通过 GUI 接口以及在命令窗口中输入相应的命令来打开和保存文件。

和大多数应用程序一样，Matlab 将当前目录作为保存和打开数据文件和 M 文件的默认目录。当然，用户可以通过 GUI 和命令窗口函数提供的目录管理工具来改变当前目录。

本章将主要讲述 Matlab 的文件和目录管理中的一些特性。

14.1 Matlab 数据文件

使用下面的 save 命令可以将当前 Matlab 工作区中的变量存储为 Matlab 自定义的数据格式：

```
>> save
```

该命令将当前 Matlab 工作区中的所有变量以 Matlab 二进制的格式存储在当前目录下的 matlab.mat 文件中。该 MAT 文件中包含的都是各变量的完整的双精度格式的值以及变量的名称。注意：MAT 文件与用户使用的操作系统平台以及 Matlab 版本均无关，在一个平台下保存的 MAT 文件完全可以在其他 Matlab 平台中打开，而不需要作任何特殊处理。

save 命令也可以将 Matlab 工作区中指定的变量保存到 matlab.mat 文件中。例如，下面的代码将变量 var1、var2 和 var3 保存到了 matlab.mat 文件中：

```
>> save var1 var2 var3
```

另外，save 命令还可以将指定的变量保存到一个用户指定的 MAT 文件中，这时需要将 MAT 文件名作为 save 命令的第一个参数传递给它。例如，下面的命令将 var1、var2 和 var3 保存在名为 filename.mat 的文件中：

```
>> save filename var1 var2 var3
```

注意：此时 filename 不要与 Matlab 工作区中的变量重名，否则 save 将执行的是将 filename、var1、var2 和 var3 四个变量存储到 matlab.mat 文件中。

由于命令-函数的二元性，上述 save 命令还可以写成如下所示的函数调用形式：

```
>> save ('filename','var1','var2','var3')
```

当函数名不是直接给出，而使用一个字符串变量保存时，只能使用上面的函数调用格式，如下面的代码所示：

```
>> fname = 'myfile';
>> save(fname,'var1','var2','var3')
```

上述命令将指定的变量 var1、var2、var3 保存在一个名为 myfile.mat 的文件中。

除了可以保存为上述的 Matlab 二进制格式，save 命令还可以将变量保存为 ASCII 文本格式，也可以将要保存的变量附加到一个已经存在的文件的后边。限于篇幅，本节不对这些特性进行详细阐述，有兴趣的读者可以参考相关的联机帮助文档。

与 save 命令相对应的就是 load 命令。这个命令用来打开由 save 命令创建的数据文件或者适用于 save 命令的数据文件。

在命令窗口中输入单个 load 命令，如：

```
>> load
```

将把在当前目录或 Matlab 的搜索路径中找到的第一个 matlab.mat 中的所有变量载入到 Matlab 工作区中。注意：这些变量会覆盖当前 Matlab 工作区中已经存在的同名变量。

要想从一个 MAT 文件中载入指定的变量，用户需要在 load 命令后面提供文件名和变量名。例如，下面的命令将从 filename 文件中载入变量 var1、var2 和 var3：

```
>> load filename var1 var2 var3
>> load('filename','var1','var2','var3')
```

其中的第二条语句是 load 命令的函数调用形式，与 save 命令一样，这种形式允许 filename 用一个字符串变量来保存。采用第二条语句的另一个方便之处在于，filename 可以是一个完整的路径字符串，这样 load 命令在载入数据时，将只在一个指定的目录中去寻找所需要的数据文件。

利用 load 命令的函数调用形式，用户还可以打开一组编号的数据文件，例如 mydata1.mat、mydata2.mat、…、mydataN.mat。请看下面的代码：

```
for i=1:N
    fname=sprintf('mydata%d',i);
    load(fname)
end
```

上述代码用 sprintf 函数在一个 For 循环中生成编号的文件名字符串，这样的一组数据文件都可以利用 load 命令载入到当前工作区中。

当用户不希望载入的变量将工作区中的同名变量覆盖，这时可以使用下面的语句将 load 载入的变量保存在一个输出参数中：

```
>> vnew = load('filename','var1','var2');
```

该命令将打开 filename.mat 文件，并将变量 var1 和 var2 载入到一个名为 vnew 的结构体变量中，该结构体变量将包含两个域，一个是 var1，一个是 var2。也就是说，vnew.var1=var1，vnew.var2=var2。

load 命令还可用于打开 ASCII 文本文件。特别是当一个数据文件是由一些 Matlab 注释行和一行行用空格隔开的数据组成时，使用下面的语句：

```
>> load filename.ext
```

可以将文件 filename.ext 中的数据载入到一个名为 filename 的双精度数据数组中。关于 load 的更详细的信息请读者参考 Matlab 的联机帮助文档。

除了 save 和 load 以外，Matlab 还提供了其他一些与文件相关的命令。例如，为了检查一个数据文件是否存在，可以使用命令 exist。例如，下面的代码检查 matlab.mat 文件是否存在于当前路径中：

```
>> exist('matlab.mat','file')
```

对于上述语句，如果 matlab.mat 文件不存在，就返回 0，如果存在，就返回 2。

如果要想知道一个文件中都包含哪些变量，可以使用 whos 命令。例如，下面的代码查看 matlab.mat 文件中都有什么变量：

```
>> whos -file matlab.mat
```

该命令将返回 matlab.mat 文件中包含的变量，其显示格式与标准 whos 命令的显示格式相同。另外，上面的语句也可以写成如下的格式：

```
>> w = whos('-file','matlab.mat')
w =
3×1 struct array with fields:
    name
    size
    bytes
    class
```

该命令将 matlab.mat 中的变量返回到一个结构体中，其各域的名字分别为：name、size、bytes、class。采用这种方法，可以将文件中的变量按照变量名、大小、占用的内存和类型保存在一个结构体中的不同域中。

最后，我们可以用命令 delete 删除一个已经存在的数据文件。例如，下面的代码删除文件 filename.ext：

```
>> delete filename.ext
```

在 Matlab 7 中，数据文件管理函数可以通过工作区浏览器以及导入向导进行访问。如果用户的工作区浏览器不可见，可以通过选择 Matlab 桌面中的 View 菜单下的 Workspace 菜单项激活它。导入向导窗口则可以通过选择 File 菜单中的 Import Data 菜单项使之出现，另外，在命令窗口中输入 uiimport 也可以激活导入向导窗口。导入向导窗口是一个多用途的 GUI，它可以使用户很方便地载入不同格式（不仅仅是 Matlab 的 MAT 文件格式）的数据文件。

14.2 数据文件的导入和导出

除了上一节讲到的 Matlab 自定义的 MAT 文件格式以及传统的 ASCII 文本格式之外，

Matlab 还支持多种标准文件格式和一些用户定义的文件格式。在这些数据文件格式中，既有图形文件格式、多媒体文件格式，还有电子表格文件格式；有的格式只能以只读方式打开，而有的格式则只能以只写方式打开。Matlab 提供的对这些数据文件的导入和导出功能使得 Matlab 能够和其他许多应用程序进行数据交换。

例如，用户可以使用图形窗口中 File 菜单下的 Save 选项将当前图形窗口中的图形保存为 Matlab 自定义的 FIG 文件格式。另外，用户还可以通过选择图形窗口中 File 菜单下的 Export 选项将当前图形窗口的图形导出为其他文件格式。上述操作也可使用 saveas 命令完成，由于用户不常用，这里不再赘述，要想获得 saveas 的帮助信息，请参见 Matlab 的在线帮助文档。

除了利用 GUI 以外，用户还可以使用 Matlab 提供的各种数据导入和导出函数完成数据的导入和导出，这些函数被列在了下面的表格中供大家参考：

数据导入和导出函数	描述
dlmread	从分隔文本文件中读入数据
dlmwrite	将数据写入分隔文本文件
textread	从文件中读入格式化文本
textscan	在利用 fopen 函数将文件打开后，再读入格式化文本
wklread	从电子表格文件中读入数据
wklwrite	将数据写入电子表格文件
xlsread	从电子表格文件读入
aviread	从 AVI 文件中读入数据
imread	从图像文件读入数据
imwrite	将数据写入图像文件
auread	从 Sun 声音文件读入数据
auwrite	将数据写入 Sun 声音文件
wavread	从 Microsoft 声音文件（.WAV 文件）读入数据
wavwrite	将数据写入 Microsoft 声音文件（.WAV 文件）
hdf	Matlab-HDF 网关函数
cdfepoch	创建用于导出通用数据文件（CDF）格式的对象
cdfinfo	获得一个 CDF 文件的信息
cdfread	从 CDF 文件中读入数据
cdfwrite	将数据写入 CDF 文件

上表中比较常用的两个函数 imread 和 imwrite 还特别提供了对多种图像文件格式的支持，包括 JPEG、TIFF、BMP、PNG、HDF、PCX 以及 XWD。要想获得表中各函数的详细信息，请参考相应的帮助文档。

另外，在命令窗口中键入 help fileformats，可以得到有关 Matlab 支持的文件格式的一个完整的列表，如下所示：

```
>> help fileformats
  Readable file formats.
```

```
Data formats                              Command     Returns
 MAT  - MATLAB workspace                  load        Variables in file.
 CSV  - Comma separated numbers           csvread     Double array.
 DAT  - Formatted text                    importdata  Double array.
 DLM  - Delimited text                    dlmread     Double array.
 TAB  - Tab separated text                dlmread     Double array.

Spreadsheet formats
 XLS  - Excel worksheet                   xlsread     Double array and cell array.
 WK1  - Lotus 123 worksheet               wk1read     Double array and cell array.

Scientific data formats
 CDF  - Common Data Format                cdfread     Cell array of CDF records
 FITS - Flexible Image                    fitsread    Primary table data
 HDF  - Hierarchical Data Format          hdfread     HDF or HDF-EOS data set

 Movie formats
 AVI  - Movie                             aviread     MATLAB movie.

Image formats
 TIFF - TIFF image                        imread      image.
 PNG  - PNG image                         imread      image.
 HDF  - HDF image                         imread      Truecolor or indexed image(s).
 BMP  - BMP image                         imread      Truecolor or indexed image.
 JPEG - JPEG image                        imread      Truecolor or grayscale image.
 GIF  - GIF image                         imread      Indexed image.
 PCX  - PCX image                         imread      Indexed image.
 XWD  - XWD image                         imread      Indexed image.
 CUR  - Cursor image                      imread      Indexed image.
 ICO  - Icon image                        imread      Indexed image.
 RAS  - Sun raster image                  imread      Truecolor or indexed.
 PBM  - PBM image                         imread      Grayscale image.
 PGM  - PGM image                         imread      Grayscale image.
 PPM  - PPM image                         imread      Truecolor image.

Audio formats
 AU   - NeXT/Sun sound                    auread      Sound data and sample rate.
 SND  - NeXT/Sun sound                    auread      Sound data and sample rate.
 WAV  - Microsoft Wave sound              wavread     Sound data and sample rate.

See also IOFUN.
```

14.3 低级文件 I/O

由于目前存在种类繁多的文件格式，Matlab 不可能为每一种文件都提供函数接口，因此 Matlab 提供了一些低级文件 I/O 函数来读取和写入二进制或格式化 ASCII 文件（因为大部分文件类型从其存储本质上讲都属于这两种类型）。我们将这些函数做了一个总结放在了下面的表格中，大家可以看到，其中的大部分函数与 ANSI C 编程语言中的文件 I/O 函数非常相像，只是函数名有个别字母的差异。实际上，其中的大多数函数都在函数体内部使用了 C 语言中的文件 I/O 函数。

类别	函数	描述/语法示例
文件打开和关闭	fopen	打开文件 fid = fopen('filenename', 'permission')
	flcose	关闭文件 status = fclose(fid)
二进制文件 I/O	fread	从一个二进制文件中读取全部或部分数据 A = fread(fid,num,precision)
	fwrite	将数据写入二进制文件 cound = fwrite(fid,array,precision)
格式化 I/O	fscanf	从文件中读取格式化数据 A = fscanf(fid,format,num)
	fprintf	将格式化数据写入文件 count = fprintf(fid,format,A)
	fgetl	从文件读取行，删除换行符 line = fget1(fid)
	fgets	从文件读取行，保留换行符 line = fgets(fid)
字符串转换	sprintf	将格式化数据写入字符串 S = sprintf(format,A)
	sscanf	在格式控制下读取字符串 A = sscanf(string,format,num)
文件定位	ferror	获取文件 I/O 状态的信息 message = ferror(fid)
	feof	检测是否到了文件结尾 TF = feof(fid)
	fseek	设置文件定位指针 status = fseek(fid,offset,origin)
	ftell	获取文件定位指针的位置 position = ftell(fid)
	frewind	将文件定位指针设置到文件开头 frewind(fid)

在上表中，fid 是一个文件标识，permission 是一个标示读写权限的字符串，包括：'r'（只读）、'w'（只写）、'a'（在文件后添加），以及'r+'（既可读又可写）。由于计算机将文本文件和二进制文件视为不同的文件类型，因此在对二进制文件进行读写操作的时候，必须在 permission 字符串后边加上'b'，例如'rb'（对二进制文件只读）。另外，表格中的 format 是一个格式字符串，它定义了用户希望的文件或数据格式，它完全遵循 ANSI 标准 C 中文件和数据格式的规范。关于这些函数的详细用法请读者参考 Matlab 在线帮助文档。

14.4 目录管理

鉴于目录管理的常用性和必要性，在 Matlab 的众多窗口中，用户希望 Matlab 能提供一个 GUI 窗口用于对当前目录及其文件的管理。Matlab 也不负众望，提供了 Current Directory 窗口来管理当前用户操作的目录及其文件。用户只需选择 Matlab 桌面窗口中 View 菜单下的 Current Directory 选项就能够显示 Current Directory 窗口。除了可以列出目录树之外，Current Directory 窗口还允许用户预览当前目录下的文件，查看它们的修改日期，搜索 M 文件中的文本，创建新的目录以及新的 M 文件，等等。鉴于对当前路径管理的重要性，Matlab 在其桌面窗口中的右键快捷菜单中也包含了 Current Directory 这一项，从而可以更方便地进入当前路径窗口。另外，用户也可以通过 Matlab 桌面窗口中提供的工具栏简单浏览当前路径和文件。

在 Matlab 6 之前，目录管理都是通过使用命令窗口函数来实现的。尽管这些函数现在已不大使用，但它们在某些特定场合还是有一定作用的。例如，如果用户要将目录和文件信息传递给一个变量，以便在 M 函数文件中进行相应的文件和目录管理，就需要使用命令窗口函数来管理目录和文件。为了使读者对 Matlab 的目录和文件管理有更全面的了解，我们将这些命令函数在下表中进行了总结。

函数	描述
cd, pwd S = cd;	显示当前的工作目录 将当前的工作目录返回给字符串 S
cd dirname	将当前的工作目录变为 dirname
copyfile(oldname,dirname) copyfile(oldname,newname)	将文件 oldname 拷贝到目录 dirname 拷贝文件 oldname 并重命名为 newname
delete filename.ext	删除文件 filename.ext
dir, ls S = dir;	显示当前目录下的文件 将目录信息返回给结构体 S
fileattrib	获取或设置文件的属性
mkdir dirname	将目录 dirname 设置为当前目录
movefile(source, destination)	将原文件或目录 source 移动到一个新的位置 destination
rmdir dirname	删除路径 dirname
what S = what;	显示当前目录中所有 Matlab 文件的有序列表 将文件列表信息返回给结构体 S
which filename S = which('filename');	显示 filename 所在的路径 将 filename 所在的路径返回给字符串 S
who who -file filename S = who('-file', 'filename');	显示工作区中的变量 显示 MAT 文件 filename.mat 中的变量 将 filename.mat 中的各变量的变量名、大小和类型返回给结构体 S

（续表）

函数	描述
whos Whos -file filename S = whos('-file', 'filename');	显示工作区中所有变量的变量名、大小和类型 显示 MAT 文件 filename.mat 中所有变量的变量名、大小和类型 将 filename.mat 中的各变量的变量名、大小和类型返回给结构体 S
Help filename S = help('filename');	在命令窗口中显示 filename 的帮助文本 将 filename 的帮助文本返回到字符串 S
Type filename	在命令窗口中显示 M 文件 filename 的具体内容

说明：上表中的这些函数大多数只需要部分路径信息就可以定位一个指定的文件，也就是说，filename 既可以包含部分或全部目录路径，也可以只是一个文件名。如果 filename 只是一个文件名，则 Matlab 就会在当前路径查找这个指定的文件，若找不到，再在 Matlab 的搜索路径中寻找。如果用户在 filename 提供了部分或全部的目录路径，Matlab 就会在用户指定的目录（包括其子目录）中寻找指定的文件。例如，如果 filename='mystuff/myfile'，Matlab 就会在 mystuff 目录和其子目录中搜索 myfile 文件。

下面的 M 函数文件 mmbytes.m 给出了上表中一些函数的用法：

```
function y=mmbytes(arg)
%MMBYTES Variable Memory Usage.
% MMBYTES and MMBYTES('base') returns the total memory in bytes
% currently used in the base workspace.
% MMBYTES('caller') returns the total memory in bytes currently
% used in the workspace where MMBYTES is called from.
% MMBYTES('global') returns the total memory in bytes currently
% used in the global workspace.
if nargin==0
      arg='base';
end
if strcmp(arg,'global')
     x=evalin('base','whos("global")');
else
     x=evalin(arg,'whos');
end
y=sum(cat(1,x.bytes));
```

上述 M 文件函数首先利用 whos 函数将 Matlab 中的变量的信息返回给一个结构体 x。然后将 x 的 bytes 域中的内容提取出来，并利用 cat 函数将其存储到一个向量中，最后，使用 sum 函数给出了这个向量的和。

由于函数 exist 用法太多，因此该函数没有出现在上面的表格中。该函数主要用于检测变量、文件和目录等是否存在。我们可以借助该函数的帮助文档来了解其众多用法，即：

```
>> help exist
  EXIST Check if variables or functions are defined.
    EXIST('A') returns:
```

```
    0 if A does not exist
    1 if A is a variable in the workspace
    2 if A is an M-file on MATLAB's search path. It also returns 2 when
        A is the full pathname to a file or when A is the name of an
        ordinary file on MATLAB's search path
    3 if A is a MEX-file on MATLAB's search path
    4 if A is a MDL-file on MATLAB's search path
    5 if A is a built-in MATLAB function
    6 if A is a P-file on MATLAB's search path
    7 if A is a directory
    8 if A is a Java class
EXIST('A') or EXIST('A.EXT') returns 2 if a file named 'A' or 'A.EXT'
exists and the extension isn't a P or MEX function extension.
EXIST('A','var') checks only for variables.
EXIST('A','builtin') checks only for built-in functions.
EXIST('A','file') checks for files or directories.
EXIST('A','dir') checks only for directories.
EXIST('A','class') checks only for Java classes.
EXIST returns 0 if the specified instance isn't found.
```

在前面的表格中，很多命令和函数都提供了将路径或文件信息返回到一个字符串中的操作，为了有助于管理这些包含了目录路径和文件名的字符串，Matlab 专门提供了几个有用的函数，我们将它们在下表中进行了总结。

函数	描述
addpath('dirname')	将目录 dirname 添加到 Matlab 的搜索路径中
[path,name,ext]=… fileparts(filename)	返回文件 filename 的路径、文件名以及文件扩展名
filesep	返回用户使用的计算机平台的文件分隔字符。文件分隔字符是用来分隔目录和文件名的字符。例如，在 PC 机中，文件分隔符是'\'。
fullfile(d1,d2,…, filename)	返回用目录树字符串 d1,d2,…表示的 filename 的完整路径和文件名
matlabroot	返回一个包含 Matlab 根目录的字符串
mexext	返回用户使用的计算机平台的 MEX 文件扩展名
pathsep	返回用户使用的计算机平台的路径分隔符。路径分隔符是用来分隔 Matlab 各级搜索路径字符串的字符。
prefdir	返回用户使用的计算机平台的 Matlab 优先目录
rmpath('dirname')	从 Matlab 搜索路径中删除目录 dirname
tempdir	返回用户使用的计算机平台的临时目录的目录名
tempname	返回用户使用的计算机平台的一个临时文件名

用户合理运用上表和前面的表格中的各个函数，可以很方便地创建像 M 文件那样功能强大的文件和目录管理函数。关于这两个表格中的每个函数更多的信息，请读者参见 Matlab 的联机帮助文档。

14.5　FTP 文件操作

目前，Matlab 7 包含了内置的 FTP 功能。在 Matlab 中，用户可以使用函数 ftp 创建一个 FTP 服务器对象，该对象与低级文件 I/O 中使用的文件标识大同小异。例如，下面的代码：

```
>> ftp_object = ftp('ftp.somewhere.org')
```

建立了一个到网址 ftp.somewhere.org 的 FTP 连接，并将该连接的标识返回给变量 ftp_object。FTP 连接建立后，用户就可以使用通用 FTP 命令来改变连接路径和执行相应的文件操作了。在这些操作过程中，用户还需要用到前面讲到的本地文件和路径的管理函数。例如，下面的代码将返回缺省路径的路径列表：

```
>> dir(ftp_object)
```

当用户执行完 FTP 操作后，可以使用 disconnect 函数关闭建立的 FTP 服务器对象。例如，下面的代码将前面创建的 ftp_object 关闭：

```
>> disconnect(ftp_object)
```

下表给出了 Matlab 提供的主要 FTP 函数：

函数	描述
ascii	将 FTP 的传输类型设为 ASCII
binary	将 FTP 的传输类型设为二进制
cd	改变当前路径
connect	打开一个超时的连接
delete	删除文件
dir	给出路径列表
disconnect	关闭连接
disp	设置显示格式
ftp	连接到一个 FTP 服务器
isconnected	检查一个连接是否有效
mget	下载文件
mkdir	创建一个新目录
mput	上载文件或目录
rename	文件重命名
rmdir	删除路径

Chapter 15

集合函数、位函数和基底函数

15.1 集合函数

从数学角度上讲，数组可以看作一系列有序排列的数值的集合。因此，在 Matlab 中，数组完全可以被看作集合进行运算。Matlab 提供了几个集合函数来对集合进行测试和比较。其中最简单的集合运算就是比较两个集合是否相等，用到的集合函数是 isequal，下面的代码给出了该函数的用法：

```
>> a = rand(2,5);  % random array
>> b = randn(2,5); % a different random array
>> isequal(a,b)    % a and b are not equal
ans =
       0
>> isequal(a,a)    % but a is certainly equal to a
ans =
       1

>> isequal(a,a(:)) % a with a as a column
ans =
      0
```

上面的结果说明，两个数组（集合）如要相等，就必须有相同的维数和相同的元素值。isequal 函数不仅仅可以用于数值型数组，还适用于所有的 Matlab 数据类型。例如，下面的代码将两个结构体变量视为集合进行运算：

```
>> a = 'a string';
>> b = 'a String';
>> isequal(a,b)  % character string equality
ans =
       0
>> a = {'one'  'two'  'three'};
>> b = {'one'  'two'  'four'};
>> isequal(a,b)  % cell array equality
ans =
       0
>> isequal(a,a)
```

```
ans =
      1
>> a.one = 'one';
>> a.two = 2;
>> a.three = pi;
>> b.two = 2;
>> b.one = 'one';
>> b.three = pi;
>> isequal(a,b) % structure equality
ans =
      1
>> isequal(a,a)
ans =
      1
```

上例说明，对于两个 Matlab 变量（包括单元数组变量），如果具有相同的维数和完全相同的元素值，则这两个 Matlab 变量就是相等的。当这两个变量为结构体变量时，只有当它们的大小相同，各域都有相同的名称和顺序，并且域中的内容也完全相同时，这两个结构体变量才是相等的。

有时候我们需要将集合中重复出现的元素删除，以保证集合中的各元素互不相等。函数 unique 可以帮助我们完成这一操作，下面给出了一个简单的例子：

```
>> a = [2:2:8;4:2:10]  % new data
a =
     2     4     6     8
     4     6     8    10
>> unique(a)            % unique elements sorted into a column
ans =
     2
     4
     6
     8
    10
```

无论原来的数组是何结构，unique 函数都返回一个经过整理后的列向量，因为 unique 函数将重复的值删除后无法保持原来的数组结构。另外，unique 函数还适用于字符串单元数组，如下例所示：

```
>> c = {'Shania' 'Britney' 'Dixie' 'Shania' 'Faith'};
>> unique(c)
ans =
    'Britney' 'Dixie' 'Faith' 'Shania'
```

该例将单元数组中重复的单元'Shania'删除。

如果我们要确定一个集合中的哪些元素是另一个集合中的成员，可以使用函数 ismember，如下例所示：

```
>> a = 1:9
a =
      1      2      3      4      5      6      7      8      9
```

```
>> b = 2:2:9
b =
       2     4     6     8
>> ismember(a,b) % which elements in a are in b
ans =
       0     1     0     1     0     1     0     1     0
>> ismember(b,a) % which elements in b are in a
ans =
       1     1     1     1
```

上例表明，当 ismember 函数接受两个向量输入时，将返回与第一个输入参数相同大小的逻辑向量，如果第一个向量的某一元素也是第二个向量的元素，则逻辑数组的对应位置将返回 True(1)，否则为 False(0)。

下面给出的是 ismember 函数用于二维数组时的情况：

```
>> A = eye(3);  % new data
>> B = ones(3);
>> ismember(A,B) % those elements in A that are also in B
ans =
      1
      0
      0
      0
      1
      0
      0
      0
      1
>> ismember(B,A)
ans =
      1
      1
      1
      1
      1
      1
      1
      1
      1
```

上例表明，当 ismember 函数接受两个二维数组输入时，将返回一个逻辑列向量，该列向量结构上与将 ismember 函数的第一个输入数组的各列按顺序连接形成的列向量相同。如果第一个数组的某一元素也是第二个数组的元素，则将在逻辑数组的对应位置返回 True(1)，否则为 False(0)。

最后，函数 ismember 还适用于字符串单元数组，如：

```
>> ismember(c,'Dixie')
ans =
     0    0    1    0    0
```

判断单元数组 c 中的哪些单元为'Dixie'。

除了上面的逻辑判断函数外，Matlab 还提供了几个集合运算函数，包括 union、intersect、setdiff 和 setxor。下面给出了这些函数简单用法的例子：

```
>> a,b            % recall prior data
a =
    1     2     3     4     5     6     7     8     9
b =
    2     4     6     8
>> union(a,b) % union of a and b
ans =
    1     2     3     4     5     6     7     8     9
>> intersect(a,b)    % intersection of a and b
ans =
    2     4     6     8
>> setxor(a,b)    % set exclusive or of a and b
ans =
    1     3     5     7     9
>> setdiff(a,b)   % values in a that are not in b
ans =
    1     3     5     7     9
>> setdiff(b,a)   % values in b that are not in a
ans =
    []
>> union(A,B,'rows') % matrix inputs produce combined rows with no repetitions
ans =
    0   0   1
    0   1   0
    1   0   0
    1   1   1
```

和前面的 isequal、unique 和 ismember 一样，上面这些集合函数同样适用于字符串单元数组，并且其用法也与这 3 个函数大同小异，读者可以通过上机练习上面的例子掌握它们的用法。

15.2　位函数

除了第 10 章讲到的数据逻辑运算外，Matlab 7 还提供了对一个无符号整数数据的各个位进行逻辑运算的函数。当然，为了兼容以前的版本，Matlab 7 也支持对浮点整数（即以双精度浮点格式存储的整数）的各个位的运算。Matlab 的位运算函数包括 bitand、bitcmp、bitor、bitxor、bitset、bitget 和 bitshift。

下面的代码给出其中一些函数的具体用法。首先我们将数据显示格式限定为 16 进制，然后查看最大的 16 位无符号整数：

```
>> format hex
>> intmax ('uint16') % largest unsigned 16-bit number
ans =
    ffff
```

之后，我们生成两个标量 a 和 b：

```
>> a = uint16(2^10-1) % first data value
a =
   03ff
>> b = uint16(567)  % second data value
b =
   0237
```

下面的代码分别执行两个整数的逐位与、逐位或、逐位异或操作：

```
>> bitand(a,b)  % (a & b)
ans =
   0237
>> bitor(a,b) % (a | b)
ans =
   03ff
>> bitxor(a,b) % xor(a,b)
ans =
   01c8
```

下面的代码将 a 的各位取反：

```
>> bitcmp(a) % complement a
ans=
   fc00
```

下面的代码获取 b 的第 7 位：

```
>> bitget(b,7) % get 7th bit of b
ans =
    0000
```

下面的代码将 b 的第 7 位设为 1：

```
>> bitset(b,7) % set 7th bit to 1
ans =
   0277
```

最后将数据显示设为默认的格式（短浮点型）。

```
>> format short g % reset display format
```

15.3 进制转换

Matlab 提供了一些实用函数用于将 10 进制数字转换成其他进制的数，转换的结果以字符串形式给出。比如，下面的代码在 10 进制和 2 进制数字之间进行转换，其中用到的转换函数是 dec2bin 和 bin2dec：

```
>> a = dec2bin(17) % find binary representation of 17
a =
    10001
>> class(a) % result is a character string
```

```
ans =
    char
>> bin2dec(a) % convert a back to decimal
ans =
    17
>> class(ans) % result is a double-precision decimal
ans =
    double
```

下面的代码则实现了 10 进制和 16 进制数字之间的转换，其中用到的转换函数是 dec2hex 和 hex2dec：

```
>> a = dec2hex(2047) % hex representation of 2047
a =
7FF
>> class(a) % result is a character string
ans =
char
>> hex2dec(a) % convert a back to decimal
ans =
        2047
>> class(ans) % result is a double precision decimal
ans =
double
```

Matlab 还提供了函数 dec2base 和 base2dec 实现 10 进制和任何进制（2~36）之间的转换。例如，下面的代码实现了 10 进制和 3 进制之间的转换：

```
>> a = dec2base(26,3)
a =
    222
>> class(a)
ans =
    char
>> base2dec(a,3)
ans =
    26
```

在 Matlab 中，36 是最多可表示的进制。这是因为，Matlab 在表示一个进制数时，需要使用 0~9 和 A~Z（共 36 个）中的一个符号来表示进制数中的一个“位”，由于这些符号只有 36 个，因此，Matlab 最多可表示 36 进制的数。

Chapter 16

时间运算

时间和日期是用户在编程时经常遇到的概念，Matlab 提供了许多函数来处理时间和日期。利用这些函数，用户可以对时间和日期进行数学运算，打印一份日历以及迅速找到指定的某一天。在 Matlab 内部，日期和时间被保存为一个双精度数（称为日期值），这个数的整数部分表示从公元 0 年 1 月 1 日到该日的天数，小数部分则表示具体的时刻。例如，2000 年 1 月 1 日午夜 0 点被表示为 730486，而该日的中午 12 点被表示为 730486.5。Matlab 采用这种表示方法是为了便于进行数学运算，但这却给用户带来了直观理解上的障碍。为此，Matlab 提供了许多函数来实现日期数字和字符串之间的转换，以及对日期和时间进行各种处理。

16.1 当前日期和时间

要获得当前的日期和时间，可以使用 clock 函数，该函数将当前的日期和时间返回到一个数组中。例如，我们可以使用下面的代码获取当前时间（由于验证的时间不同，用户得出的结果会与此不同）：

```
>> T = clock
T =
        2005        11        3        21        59        11.7
```

当用 clock 查看当前时间时，其返回值 T 的各元素代表的时间单位分别为：T=[year month day hour minute seconds]。因此，我们获取的当前时间为 2005 年 11 月 3 日 21 点 59 分 11.7 秒。

与 clock 不同，函数 now 将返回当天的日期值，如下例所示：

```
>> format long

>> t = now
t =
    7.326199174151736e+005

>> format short g
```

对于 Matlab 而言，上述的 T 和 t 分别是相同时间信息的不同表示方法。

另外，函数 date 以 dd-mmm-yyyy 的格式返回表示当天日期的字符串，如下例所示：

```
>> date
ans =
03-Nov-2005
```

16.2 日期格式转换

一般而言，时间运算都遵循以下步骤：首先将时间转换成日期值，然后针对该日期值进行标准数学运算，最后将运算结果转换成用户期望的日期格式。因此，在时间运算中，日期格式转换是至关重要的一环。Matlab 支持以下 3 种日期格式：①双精度日期数字（日期值）。②各类日期字符串。③数值日期向量（如 clock 函数的返回值 [year,month,day,hour,minute,seconds]）。

函数 datestr 用于将日期值转换成一个表示日期的字符串。datestr 的调用语法为：datestr (date, dateform)，其中 date 为要转换的日期值，dateform 为日期的字符串类型，datestr 的帮助文档给出了该参数的详细描述：

```
>> help datestr
 DATESTR String representation of date.
     DATESTR(D,DATEFORM) converts D, a data vector(as returned by
     DATEVEC) or serial date number(as returned by DATENUM), or a free format
     date string into a date string with a format specified by format number
     or string DATEFORM (see table below). By default, DATEFORM is 1, 16,
     or 0 depending on whether D contains dates, times or both. Date strings
     with 2 character years are interpreted to be within the 100 years centered
     around the current year.

     DATESTR(D,DATEFORM,PIVOTYEAR) uses the specified pivot year as the
     starting year of the 100-year range in which a two-character year
     resides. The default pivot year is the current year minus 50 years.
     DATEFORM = -1 uses the default format.
     DATEFORM number    DATEFORM string         Example
        0               'dd-mmm-yyyy HH:MM:SS'  01-Mar-2000 15:45:17
        1               'dd-mmm-yyyy'           01-Mar-2000
        2               'mm/dd/yy'              03/01/00
        3               'mmm'                   Mar
        4               'm'                     M
        5               'mm'                    03
        6               'mm/dd'                 03/01
        7               'dd'                    01
        8               'ddd'                   Wed
        9               'd'                     W
       10               'yyyy'                  2000
       11               'yy'                    00
       12               'mmmyy'                 Mar00
       13               'HH:MM:SS'              15:45:17
       14               'HH:MM:SS PM'           3:45:17 PM
       15               'HH:MM'                 15:45
```

```
        16                'HH:MM PM'              3:45 PM
        17                'QQ-YY'                 Q1-96
        18                'QQ'                    Q1
        19                'dd/mm'                 01/03
        20                'dd/mm/yy'              01/03/00
        21                'mmm.dd,yyyy HH:MM:SS'  Mar.01,2000 15:45:17
        22                'mmm.dd,yyyy'           Mar.01,2000
        23                'mm/dd/yyyy'            03/01/2000
        24                'dd/mm/yyyy'            01/03/2000
        25                'yy/mm/dd'              00/03/01
        26                'yyyy/mm/dd'            2000/03/01
        27                'QQ-YYYY'               Q1-1996
        28                'mmmyyyy'               Mar2000
        29(ISO 8601)      'yyyy-mm-dd'            2000-03-01
        30(ISO 8601)      'yyyymmddTHHMMSS'       20000301T154517
        31                'yyyy-mm-dd HH:MM:SS'   2000-03-01 15:45:17
    See also DATE, DATENUM, DATEVEC, DATETICK.
```

下面给出了 datestr 函数的几个调用实例：

```
>> t = now
t =
    7.326199203191898e+005
>> datestr(t)
ans =
03-Nov-2005 22:05:16

>> class(ans)
ans=
char
>> datestr(t,12)
ans =
 Nov05
>> datestr(t,23)
ans =
11/03/2005
>> datestr(t,25)
ans =
 05/11/03
>> datestr(t,13)
ans =
22:05:16
>> datestr(t,29)
ans =
2005-11-03
```

与 datestr 相反，函数将一个日期字符串转换成日期值。datenum 函数的调用格式为 datenum(str)，其中 str 为要转换的日期字符串。另外，它还可以将以数字表示的日期转换成日期值，其调用格式有两种：datenum(year,month,day)和 datenum(year,month,day,hour, minute, second)。

下面的例子演示了 datenum 的具体用法：

```
>> t = now
t =
    7.326199217818634e+005
>> ts = datestr(t)
ts =
03-Nov-2005 22:07:22
>> datenum(ts)
ans =
     7.326199217824074e+005
>> datenum(2004,5,14,16,48,07)
ans =
     7.320817000810185e+005
>> datenum(2004,5,14)
ans=
     732081
```

函数 datevec 用于将 datestr 中指定的格式 0，1，2，6，13，14，15 或 16 的日期字符串转换为一个仅包含日期分量的数值向量。另外，该函数还可将一个日期值转换成仅包含日期分量的数值向量。下面的代码演示了该函数的用法：

```
>> c = datevec('12/24/1984')
c =
     1984     12     24     0     0     0
>> [yr,mo,day,hr,min,sec] = datevec('24-Dec-1984 08:22')
yr =
     1984
mo =
     12
day =
     24
hr =
     8
min =
     22
sec =
     0
```

16.3 日期函数

日期函数用于从一个日期值或日期字符串中找出具体的日子或星期。例如，要想知道某天是星期几，可以用函数 weekday，该函数根据一个日期字符串或日期值返回该日的星期数。注意：Matlab 的星期数计算是按照西方的习惯进行的，即星期天为一星期的第一天，星期六为第 7 天。下面的代码给出了 weekday 的具体用法：

```
>> [d,w] = weekday(728647)
d =
    2
w =
    Mon

>> [d,w] = weekday('21-Dec-1994')
```

```
d =
    4
w =
    Web
```

要想知道任何一个月的最后一天是几号，用户可以使用 eomday 函数。由于闰年的存在，因此该函数需要年和月两个输入参数。下面的代码演示了该函数的应用：

```
>> eomday(1996,2) % divisible by 4 is a leap year
ans =
    29

>> eomday(1900,2) % divisible by 100 not a leap year
ans =
    28

>> eomday(2000,2) % divisible by 400 is a leap year
ans =
    29
```

利用 calendar 函数，用户可以生成指定月份的一个日历，该日历既可以显示在命令窗口中，又可以存储到一个 6×7 的数组中。下面的代码演示了 calendar 函数的用法：

```
>> calendar(date)
                        Feb 2004
        S       M      Tu       W      Th       F       S
        1       2       3       4       5       6       7
        8       9      10      11      12      13      14
       15      16      17      18      19      20      21
       22      23      24      25      26      27      28
       29       0       0       0       0       0       0
        0       0       0       0       0       0       0
>> calendar(1954,9)
                        Sep 1954
        S       M      Tu       W      Th       F       S
        0       0       0       1       2       3       4
        5       6       7       8       9      10      11
       12      13      14      15      16      17      18
       19      20      21      22      23      24      25
       26      27      28      29      30       0       0
        0       0       0       0       0       0       0
>> x = calendar(2004,2)
x =
        1       2       3       4       5       6       7
        8       9      10      11      12      13      14
       15      16      17      18      19      20      21
       22      23      24      25      26      27      28
       29       0       0       0       0       0       0
        0       0       0       0       0       0       0
>> class(x)
ans =
double
```

16.4 计时函数

当用户需要计算一组 Matlab 操作的运行时间时，可以使用 tic 和 toc 函数。tic 函数启动一个秒表，表示计时开始；toc 则停止这个秒表，表示计时结束，并计算出所经历的时间（单位为秒）。下面的代码连续两次计算 plot(rand(50,5))这条指令的执行时间：

```
>> tic; plot(rand(50,5)); toc
elapsed_time =
    0.33

>> tic; plot(rand(50,5)); toc
elapsed_time =
    0.11
```

读者会发现两条同样的 plot 命令在计算时间上的差别。第二条 plot 命令要比第一条执行得快，这是因为 Matlab 已经在执行第一条 plot 命令时生成了 Figure 窗口并且已经将所需要的函数编译到了内存，这样第二条指令就省去了创建 Figure 窗口以及函数搜索和编译的时间。

除了 tic 和 toc 外，Matlab 还提供了两个函数 cputime 和 etime，用来计算一次运算所占用的时间。其中，函数 cputime 返回以秒为单位的、自当前 Matlab 程序段启动之后到调用该函数所占用的 CPU 时间；函数 etime 计算两个以 6 元素行向量格式（例如函数 clock 与 datevec 的返回值）表示的时间向量之间的时间间隔。实际上，函数 tic 和 toc 内部也在利用 clock 和 etime 进行计时。下面的这些代码演示了 cputime 和 etime 的用法，其中 myoperation 是一个用户自定义的 M 脚本文件：

```
>> t0 = cputime; myoperation; cputime-t0
ans =
    0.149999999999991

>> t1 = clock; myoperation; etime(clock,t1)
ans =
    11.284853
```

16.5 图形的时间标签

很多时候，我们在用 Matlab 作图时都需要用日期或时间字符串作某一坐标轴的标签。使用函数 datetick 可自动为一个坐标轴设置时间标签。注意，必须首先用 plot 函数画好一个随时间变化的曲线，然后才能用 datetick 函数为时间坐标轴设置希望的格式。例如，下边的例子画出了某地区从 1900 年到 1990 年每十年一次的人口统计（单位：万人）：

```
>> t = (1900:10:1990)';
>> p = [ 75.995; 91.972; 105.711; 123.203; 131.669;
        150.697; 179.323; 203.212; 226.505; 249.633];
>> plot(datenum(t,1,1),p)
>> datetick('x','yyyy')          % use 4-digit year on the x-axis
>> title('Figure 16.1: Population by Year')
```

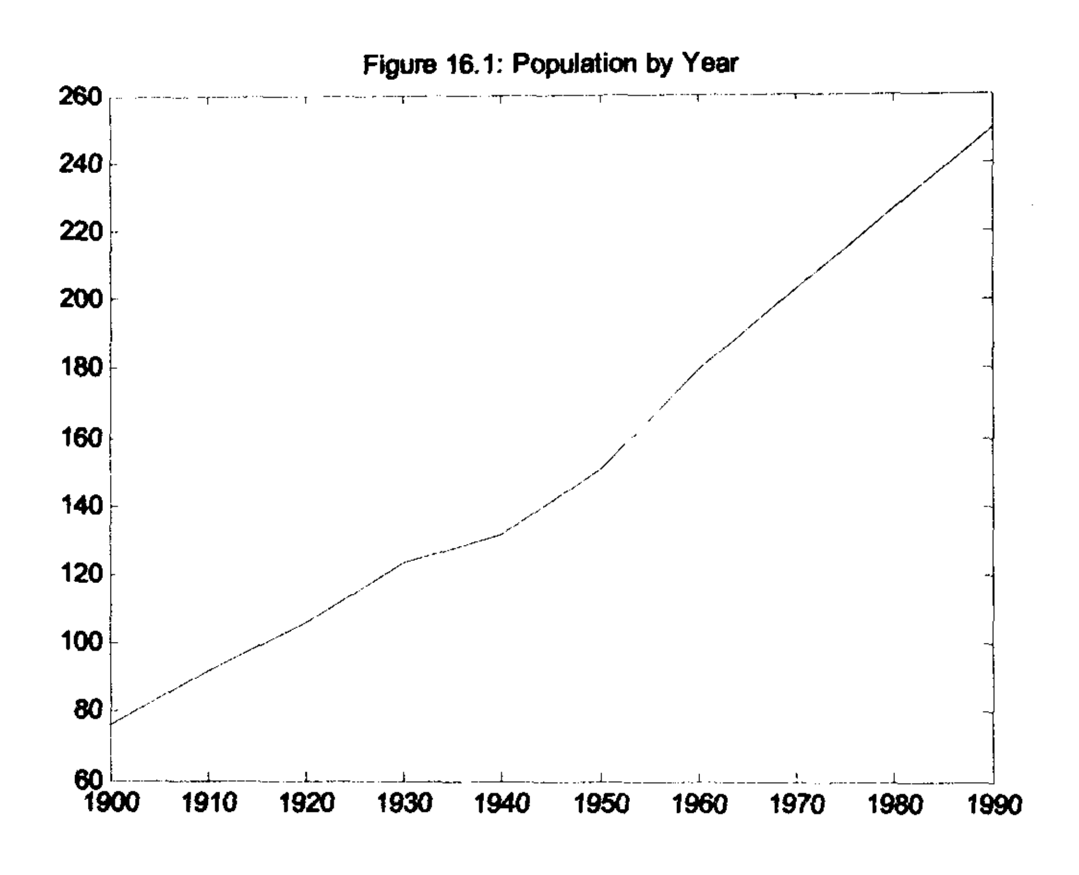


图 16.1 人口年度统计图

为了强化理解，我们再看一个某公司从 1998 年 11 月到 1999 年 10 月的销售情况的例子，该例画出了每月销售额（单位：百万美元）的柱状统计图，下面是具体实现的代码：

```
>> y = [1998 1998 1999*ones(1,12)]';
>> m = [11 12 (1:12)]';
>> s = [1.1 1.3 1.2 1.4 1.6 1.5 1.7 1.6 1.8 1.3 1.9 1.7 1.6 1.95]';
>> bar(datenum(y,m,1),s)

>> datetick('x','mmmyy')
>> ylabel('$ Million')
>> title('Figure 16.2: Monthly Sales')
```

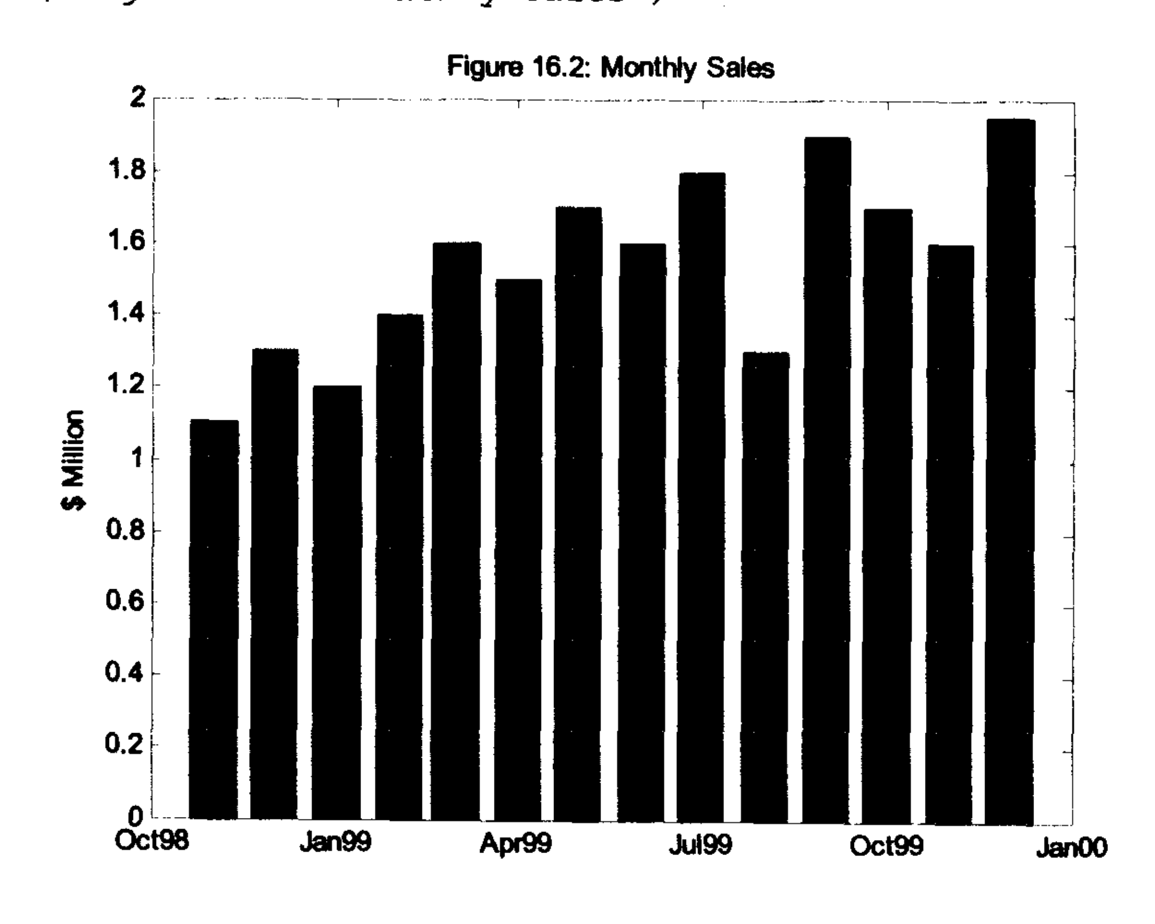


图 16.2 月度销售统计图

Chapter 17

矩 阵 代 数

Matlab 的最初设计目的就是为程序员或科研人员编写专业化的数值线性代数程序提供一个简单实用的接口。随着 Matlab 版本的不断升级，Matlab 提供了一些更贴近用户的特性，比如结果可视化和图形和图形用户界面，就使得数字线性代数程序逐渐淡出了用户视野。然而，从本质上讲，矩阵仍旧是 Matlab 的核心，因此，Matlab 还是提供了大量的矩阵代数函数来处理和操作矩阵。

请大家注意，虽然 Matlab 也支持多维数组，但矩阵代数却是只涉及到一维和二维数组，就是前面章节讲到的向量和矩阵。

17.1 线性方程组

线性代数中最常见的一类问题就是线性方程组求解。例如，我们看下面这一组等式：

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 366 \\ 804 \\ 351 \end{bmatrix}$$

$$\mathbf{A \cdot x = y}$$

上面的等式和简化表达式都是从数学角度定义的，其中，点号（•）表示矩阵乘法，这与数组的逐元素乘法是不同的。在 Matlab 中，矩阵乘法用星号（*）表示。线性代数中的一个基本问题是验证上述等式的解是否存在，当有解时，如何求解。在代数理论中，有好几种方法求上述等式的解，如高斯消元法、LU 因数分解法，或者直接用 A^{-1} 求解。本书并不对这些矩阵代数理论和方法作过多阐述，只是向读者演示如何使用 Matlab 来解决上述问题。

下面我们根据前面给出的例子，来看一下线性方程组求解问题。首先，必须创建两个数组 A 和 y：

```
>> A = [1 2 3; 4 5 6
7 8 0 ]
A =
     1         2         3
```

```
    4         5         6
    7         8         0
>> y = [366;804;351]
y =
    366
    804
    351
```

A 之所以采用上述的输入方法，是为了向读者展示在输入数组时两种不同的区分分隔行的方法：分号分隔法（如 A 的第 1、2 行之间）和回车换行法（如 A 的第 2、3 行之间）。

由线性代数理论可知，只有在 A 的秩以及广义矩阵[A y]的秩都等于 3 时，上述等式才有一个惟一的解。另外，用户还可以检查 A 的条件数。如果条件数不是特别大，那么 A 就有一个便于处理的逆存在。下面的代码分别计算 A 和[A y]的秩，以及 A 的条件数：

```
>> rank(A)
ans =
    3
>> rank([A y])
ans =
    3

>> cond(A)  % close to one is best
ans =
    35.106
```

从上面的结果可以看出，等式 A • x = y 是存在解的。Matlab 提供了两种方法完成求解过程。一种方法是直接利用 A 的逆 A^{-1} 求解，即解为 $x=A^{-1}\cdot y$。下面的代码给出了求解结果：

```
>> x = inv(A)*y     % Avoid this approach if possible
x =
        25
        22
        99
```

上例中，inv(A)是一个 Matlab 函数，它用于计算 A^{-1}；星号（*）表示矩阵乘法运算。另外一种方法是利用矩阵左除运算符（\）求解，即 $x = A\backslash y$。下面的代码给出了求解结果：

```
>> x = A\y  % Recommended approach to solving sets of equations
x =
        25
        22
        99
```

上式中的 \ 表明执行的是矩阵左除运算，该运算用到了 LU 因式分解法，LU 因式分解法的具体原理请读者参考有关线性代数方面的书籍。注意，\（包括其他的运算符如+、-、*、/、^等）前面有无点号代表的运算截然不同，当前面没有点号时，表明执行的是一个矩阵运算；当前面有点号时，表明执行的是一个元素对元素的数组运算。

尽管 Matlab 提供了两种方法，但我们更倾向于第二种方法，因为该方法用到的浮点数运算较少，执行速度相对较快，另外，这种方法由于采用了 LU 因式分解，因此得出的结果要精确得多（由于前面的例子只是一个最简单的例子，因此两种方法得出了相同的结果，

但如果要解决的问题非常复杂时，矩阵左除法的优势就比较明显）。最后，当用这两种方法求解时，如果 Matlab 找不到正确的解，就显示出一条报警信息。

对前面的方程组 A • x = y 进行转置，即(A•x)'=y'，则可以得到 x'•A'=y'，那么 x'和 y'就变成了行向量。因此，在表示一个线性方程组时，利用行向量和利用列向量进行表示是等效的。为了与列向量表示的方程组相区别，我们令 x'=u，A'=B，y'=v，则 x'•A'=y'可写作 u•B=v。在 Matlab 中，采用行向量表示的方程组的求解也有两种方法：逆矩阵法（即 $u=B^{-1}v$）和矩阵右除法（即 u=v/B）。

有一点值得说明，当 Matlab 进行矩阵左除（\）或矩阵右除（/）时，会根据系数矩阵（A 或 B）的结构特征来决定采取什么样的内部算法进行求解。特别地，如果矩阵是一个上三角或下三角矩阵，Matlab 就不会对这个矩阵进行因式分解而直接进行必要的左除或右除运算。总之，Matlab 的根本目的是：选择尽可能快的速度求出正确的结果。

当用户事先知道系数矩阵的结构时，可以使用函数 linsolve 求线性方程组的解。linsolve 函数的调用格式为：linsolve(A,y,options)，其中 A 为系数矩阵，y 为解，options 为矩阵的结构特征，它是一个由逻辑值（True/False）组成的结构体。利用函数 linsolve 求解，Matlab 就不需要花费时间分析矩阵的结构，从而使求解的速度得以提高。

根据线性代数原理，当等式的数量和未知数的数量不相等的时候，通常方程组并不存在一个惟一的解，要想得到一个符合实际应用的解，用户必须给该方程组附加一定的约束条件。在方程组 A • x = y 中，如果 rank(A)=min(r,c)，其中 r 和 c 分别是矩阵 A 的行数和列数，并且等式的数量多于未知数的数量（即 r>c）时，该方程称为超定方程。在 Matlab 中，超定方程的解是使用矩阵左除（\）或矩阵右除（/）得到的平均误差 e=A•x−y 最小的解，该解被称为最小二乘解。下面给出了一个求超定方程组解的例子：

```
>> A = [1 2 3;4 5 6;7 8 0;2 5 8]  % 4 equations in 3 unknowns
A =
        1       2       3
        4       5       6
        7       8       0
        2       5       8
>> y = [366 804 351 514]'  % a new r.h.s. vector
y =
    366
    804
    351
    514
>> x = A\y    % least squares solution
x =
         247.98
        -173.11
         114.93
>> e = A*x-y  % this residual has the smallest norm.
e =
   -119.4545
     11.9455
      0.0000
```

```
     35.8364
>> norm(e)
ans =
   125.2850
```

除了可以用左除和右除运算求最小二乘解之外，Matlab 还提供了函数 lscov 和 lsqnonneg 来求超定方程组的解。lscov 函数主要用于解决数据的协方差矩阵（或加权矩阵）已知情况下的加权最小二乘问题，而 lsqnonneg 函数给出的是非负最小二乘解，也就是说所有的解都必须为正数。

与超定方程组相对应的是欠定方程组。它指等式的数量少于未知数的数量（即 r<c）的方程组。欠定方程组存在无穷多个解，利用 Matlab 可以求出其中的两个解。一个解利用除法得到，该解在所有的解中 0 元素最多；另一个解通过计算 x=pinv(A)*y 得到，该解的长度（也叫范数）小于所有其他的解，因此又叫最小范数解。下例给出了求这两个解的实例代码：

```
>> A = A'  % create 3 equations in 4 unknowns
A =
       1     4     7     2
       2     5     8     5
       3     6     0     8
>> y = y(1:3) % new r.h.s. vector
y =
     366
     804
     351
>> x = A\y  % solution with maximum zero elements
x =
          0
  -165.9000
    99.0000
   168.3000
>> xn = pinv(A)*b % minimum norm solution
xn =
    30.8182
  -168.9818
    99.0000
   159.0545
>> norm(x)  % norm of solution with zero elements
ans=
   256.2200
>> norm(xn) % minimum norm solution has smaller norm!
ans=
   254.1731
```

其中的 pinv 称为伪逆函数，norm 则返回向量或矩阵的范数。

17.2 矩阵函数

除了可以对线性方程组求解外，Matlab 还提供了其他一些矩阵函数用于求解另外的数值线性代数问题。从总体上讲，Matlab 提供了处理几乎所有常见和不常见的数值线性代数问题的函数。限于篇幅，我们不可能对这些函数一一介绍，而是选择了一些最常用的函数在下边的表格中进行了简要描述（如无特别声明，下表中函数参数中的大写字母均为矩阵，小写字母则为向量或标量，线性方程组均为 Ax=y）：

函数	描述
/ 和 \	矩阵的左除和右除，用于求解线性方程组
accumarray(ind,val)	利用累加创建数组
A^n	求 A 的 n 次幂，例如 A^3=A*A*A
balance(A)	将 A 进行缩放以提高其特征值精度
[V,D]=cdf2rdf(V,D)	将复数对角矩阵转化为两个实数对角矩阵
chol(A)	将 A 进行 Cholesky 因式分解
cholinc(A,DropTol) cholinc(A,Options)	将 A 进行不完全 Cholesky 因式分解
cholupdate(R,X)	Cholesky 因式分解的秩 1 升级
cond(A)	利用奇异值分解求 A 的条件数
condest(A)	求 A 的范数 1 的条件数估计
[V,D,s]=condeig(A)	求 A 的与重复特征值相对应的条件数
det(A)	求矩阵的行列式
dmperm(A)	对 A 进行 Dulmage-Mendelsohn 排列
eig(A)	求矩阵的特征值和特征向量
[V,D]=eig(A)	求 A 的特征向量矩阵（V）和特征值对角矩阵（D）
expm(A)	矩阵指数函数
funm(A)	矩阵通用函数
gsvd(A,B)	求 A 的广义奇异值
[U,V,X,C,S]=gsvd(A)	对 A 进行广义奇异值分解
hess(A)	求 A 的 Hessenburg 标准型
inv(A)	求 A 的逆
linsolve(A,y,options)	快速求解 Ax=y，其中 A 的结构由 options 给定
logm(A)	矩阵对数运算
lscov(A,y,V)	已知数据的协方差矩阵（V），求线性方程组的最小二乘解
lsqnonneg(A,y)	求线性方程组的非负最小二乘解
[L,U,P]=lu(A)	对矩阵 A 进行 LU 分解
minres(A,y)	利用最小残差法求线性方程组的解

（续表）

函数	描述
norm(A,type)	求矩阵或向量（由 type 指定）的范数
null(A)	求 A 的零空间
orth(A)	求 A 的正交空间
pinv(A)	求 A 的伪逆矩阵
planerot(X)	对 X 进行平面旋转
poly(A)	求 A 的特征多项式
polyeig(A0,A1,…)	多项式的特征值解
polyvalm(A)	求 A 的矩阵多项式
qr(A)	对 A 进行正交三角分解
qrdelete(Q,R,J)	从 QR 分解中删除行或列
qrinsert(Q,R,J,X)	在 QR 分解中插入行或列
qrupdate(Q,R,U,V)	Cholesky 因式分解的秩 1 升级
qz(A,B)	广义特征值问题求解
rank(A)	利用奇异值分解求 A 的秩
rcond(A)	对 A 进行 LAPACK 倒数条件估计
rref(A)	将矩阵 A 变换为行阶梯型
rsf2csf(A)	将 A 由实块对角阵转换成复块对角阵
schur(A)	对 A 进行 Schur 分解
sqrtm(A)	求 A 的平方根
subspace(A,B)	求两个子空间 A 和 B 之间的角度
svd(A)	求矩阵 A 的奇异值
[U,S,V]=svd(A)	对 A 进行奇异值分解
trace(A)	求矩阵 A 的迹（即对角元素的和）

17.3 特殊矩阵

为了满足广泛的需要，Matlab 还提供了一系列特殊矩阵，这些矩阵有的有着广泛的用途，有的则是为解决某一特定问题定义的。下表给出了这些矩阵的表示方法及描述，要知道有关这些矩阵更多的信息，请参看 Matlab 的联机帮助文档。

矩阵	描述
[ ]	空矩阵
blkdiag(a0,a1,…)	以输入参数为对角元素生成对角矩阵
compan(P)	求多项式的伴随矩阵
eye(r,c)	产生 r 行 c 列的单位阵
gallery	生成一系列测试矩阵（50 个以上）

（续表）

矩阵	描述
hadamard(n)	生成一个 n 阶的 Hadamard 矩阵
hankel(C)	生成 C 的 Hankel 矩阵
hilb(n)	生成 n 阶的 Hilbert 矩阵
invhilb(n)	生成 n 阶的逆 Hilbert 矩阵
magic(n)	生成 n 阶的魔幻矩阵
ones(r,c)	生成一个 r 行 c 列的全 1 矩阵
pascal(n)	生成 n 阶的 Pascal 矩阵
rand(r,c)	生成一个 r 行 c 列的随机矩阵（元素值介于 0 和 1 之间）
randn(r,c)	生成一个 r 行 c 列的零均值和单位方差的正态分布的随机矩阵
rosser	典型的对称矩阵特征值问题测试
toeplitz(C,R)	生成 Toeplitz 矩阵
vander(C)	生成 Vandermonde 矩阵
wilkinson(n)	生成 n 阶的 Wilkinson 特征值测试矩阵
zeros(r、c)	生成一个 r 行 c 列的全 0 矩阵

17.4 稀疏矩阵

在很多实际应用中，用户往往会遇到只有少数元素为非 0 值的矩阵，我们称这些矩阵为稀疏矩阵。例如，电路仿真和有限元分析程序所处理的矩阵中通常包含的非 0 元素个数往往不及元素总数的 1%！如果一个矩阵很大，比如维数大于 100，并且 0 元素所占的比例很大，让计算机去存储这些 0 元素是很浪费内存空间的，并且总是对这些 0 元素进行数学运算也是十分耗时的。为了更有效地存储和处理稀疏矩阵，人们对其采用了一系列优化技术：为了避免对 0 元素的存储，通常只存储稀疏矩阵中的非 0 元素，并用行索引和列索引标明每一个非 0 元素在原矩阵中的位置；同样，为了避免对 0 元素进行数学运算，人们也采取了一些专门的算法来处理稀疏矩阵，比如在求解线性方程组的过程中，尽量减少对 0 元素的运算，并且最大限度地减少中间结果中的非 0 元素。

稀疏矩阵的优化技术无论从理论上还是实现上都不是简单的事情。不过对于 Matab 用户而言，处理稀疏矩阵将变得十分容易，因为 Matlab 将稀疏矩阵优化的复杂性隐藏了起来，用户只需要利用 Matlab 提供的函数接口进行调用即可。在 Matlab 中，稀疏矩阵也可以像普通矩阵一样被存储在一个变量中，对普通矩阵进行处理的大部分命令和函数都适用于稀疏矩阵。例如，假设 s 为稀疏矩阵，s(i,j)=1 表示将稀疏矩阵 s 的第 i 行第 j 列的元素置为 1。

限于篇幅原因，本书仅讨论了稀疏矩阵的创建和怎样实现普通矩阵和稀疏矩阵之间的转换。一般而言，普通矩阵和普通矩阵之间进行运算的结果仍是普通矩阵，稀疏矩阵和稀疏矩阵之间进行运算的结果也仍是稀疏矩阵。当普通矩阵和稀疏矩阵之间进行混合运算时，Matlab 通常根据结果矩阵中非零元素的个数决定得出稀疏矩阵还普通矩阵，但一般情况下将返回稀疏矩阵。

在 Matlab 中创建稀疏矩阵很简单，只要调用函数 sparse 即可，例如，下列创建了一个 10×10 的单位矩阵，并用稀疏矩阵表示：

```
>> As = sparse(1:10,1:10,ones(1,10))
As =
    (1,1)        1
    (2,2)        1
    (3,3)        1
    (4,4)        1
    (5,5)        1
    (6,6)        1
    (7,7)        1
    (8,8)        1
    (9,9)        1
    (10,10)      1
```

如上例所示，sparse 函数的调用语法为：As=sparse (i,j,s)，其中 i、j、s 必须为相同长度的向量。用该语法创建的稀疏矩阵的非 0 元素 s(k)出现在矩阵的第 i(k)行第 j(k)列。请注意，稀疏矩阵在显示时，只显示非 0 元素及其所在的行和列的位置。

我们也可以利用矩阵转换的方式创建稀疏矩阵，比如，上面的稀疏矩阵也可以利用下面的代码创建：

```
>> As = sparse(eye(10))
As =
    (1,1)        1
    (2,2)        1
    (3,3)        1
    (4,4)        1
    (5,5)        1
    (6,6)        1
    (7,7)        1
    (8,8)        1
    (9,9)        1
    (10,10)      1
```

该命令首先生成一个 10×10 的单位阵，然后将其作为 sparse 的输入参数，从而创建稀疏矩阵 As。采用矩阵转换的办法创建稀疏矩阵相对来说比较耗费内存，因为它进行的是对矩阵的操作，因此在实际中很少采用。

要想使一个稀疏矩阵在屏幕上完整显示，可以使用 full 函数将其生成等值的普通完整矩阵，然后再显示。例如，我们将 As 变换成完整的普通矩阵，代码如下：

```
>> A = full(As)
A = 1     0     0     0     0     0     0     0     0     0
    0     1     0     0     0     0     0     0     0     0
    0     0     1     0     0     0     0     0     0     0
    0     0     0     1     0     0     0     0     0     0
    0     0     0     0     1     0     0     0     0     0
    0     0     0     0     0     1     0     0     0     0
    0     0     0     0     0     0     1     0     0     0
```

```
  0    0    0    0    0    0    0    1    0    0
  0    0    0    0    0    0    0    0    1    0
  0    0    0    0    0    0    0    0    0    1
```

下面的代码向读者证明了稀疏矩阵较之普通矩阵在存储上的优势：

```
>> B = eye(200);
>> Bs = sparse(B);
>> whos
  Name      Size          Bytes   Class
  B        200×200        320000  double array
  Bs       200×200          3204  sparse array
Grand total is 40200 elements using 323204 bytes
```

在上面的代码中，由于一个200×200的单位阵只包含了0.5%的非0元素，因此，采用稀疏矩阵进行存储只需要3204个字节，而采用普通矩阵存储需要的内存量几乎是稀疏矩阵的100倍！

17.5 稀疏矩阵函数

鉴于稀疏函数在数据存储上的巨大优势，Matlab提供了许多函数描述稀疏函数的特性，对稀疏函数进行优化以及利用稀疏函数进行实际问题求解等。本节不打算对这些函数进行一一阐述，而将这些函数列在下表中供读者参考。

稀疏矩阵函数	描述
bicg	求双共轭梯度迭代线性方程的解
bicgstab	求双共轭梯度稳定迭代线性方程的解
cgs	求二次共轭梯度迭代线性方程的解
cholinc	不完全Cholesky分解
colamd	列估计最小度重排序方法
colamdtree	后边带了列消元树排序后的colamd方法
colmmd	列最小度排序
colperm	对列进行随机排序
condest	1范数估计
dmperm	Dulmage-Mendelsohn重排序方法
eigs	使用了ARPACK的特征值
etree	矩阵消元树结构
etreeplot	绘制消元路径
find	寻找非0元素的索引
full	将稀疏矩阵转换成完整矩阵
gmres	求广义最小残差迭代线性方程的解
gplot	绘制图论图形

（续表）

稀疏矩阵函数	描述
issparse	判断是否为稀疏矩阵，若是，返回 True，否则返回 False
lsqr	标准方程中的共轭梯度的 LSQR 实现
luinc	不完全 LU 因式分解
minres	最小残差迭代线性方程求解
nnz	求矩阵中非 0 元素的个数
nonzeros	提取矩阵中的非 0 元素
normest	矩阵的 2 阶范数估计
nzmax	求分配给非 0 元素的存储空间
pcg	求预处理共轭梯度迭代线性方程组的解
qmr	Quasi 最小残差迭代线性方程组解法
randperm	产生随机排列的数组
spalloc	为稀疏矩阵分配内存空间
sparse	创建稀疏矩阵
spaugment	建立最小二乘增广系统
spconvert	从外部格式中载入稀疏矩阵
spdiags	利用对角元素生成稀疏矩阵
speye	单位稀疏矩阵
spfun	将一个函数应用于非 0 元素
spones	将所有的非 0 元素用 1 替换
spparms	设置稀疏矩阵程序的参数
sprand	创建均匀分布的随机稀疏矩阵
sprandn	创建高斯分布的随机稀疏矩阵
sprandsym	创建对称的随机稀疏矩阵
sprank	求结构秩的值
spy	稀疏矩阵的图形表示
svds	产生一些奇异值
symbfact	符号因式分解
symamd	对称估计最小阶次重排序法
symamdtree	后边带了对称消元树排序后的 symamd
symmd	对称最小阶次重排序
symmlq	对称 LQ 迭代线性方程的求解
symrcm	对称的反向 Cuthill-Mckee 重排序
treelayout	变换成树状结构
treeplot	画出树状图

Chapter 18

数 据 分 析

Matlab 内在的面向数组的特性使得 Matlab 在对数据进行分析时显得游刃有余。在默认情况下，Matlab 将数据集视为按列存储的数组，实际上，Matlab 可以对任意指定方向的数据进行分析。

> 如不特别声明，一个数组的每一列代表不同的观测变量，每一行则代表该变量的一次采样或观测值。

18.1 基本统计分析

我们先来看一个统计分析的简单例子。假如我们在考察完 3 个城市某月（31 天）的日最高气温（摄氏度）后，将观测值记录下来并保存在一个 M 脚本文件中的变量 temps 中。现在我们在命令窗口中运行这个 M 脚本文件，这样 M 脚本文件中的变量就被载入到了 Matlab 工作区中。首先，我们在命令窗口输入变量 temps，来查看其中的观测值，如下面的代码所示：

```
>> temps
temps =
    12     8    18
    15     9    22
    12     5    19
    14     8    23
    12     6    22
    11     9    19
    15     9    15
     8    10    20
    19     7    18
    12     7    18
    14    10    19
    11     8    17
     9     7    23
     8     8    19
    15     8    18
```

```
     8       9      20
    10       7      17
    12       7      22
     9       8      19
    12       8      21
    12       8      20
    10       9      17
    13      12      18
     9      10      20
    10       6      22
    14       7      21
    12       5      22
    13       7      18
    15      10      23
    13      11      24
    12      12      22
```

在 temps 中，每行包含的是 3 个城市在某一天的最高气温，每列包含的是不同城市一个月来每天的最高气温。为了使用户能够看到数据的变化规律，我们可以将这些数据图形化，即用下面的语句绘制出每个城市该月日最高气温的变化曲线：

```
>> d =1:31;          % number the days of the month
>> plot(d,temps)

>> xlabel('Day of Month'),ylabel('Celsius')
>> title('Figure 18.1: Daily High Temperatures in Three Cities')
```

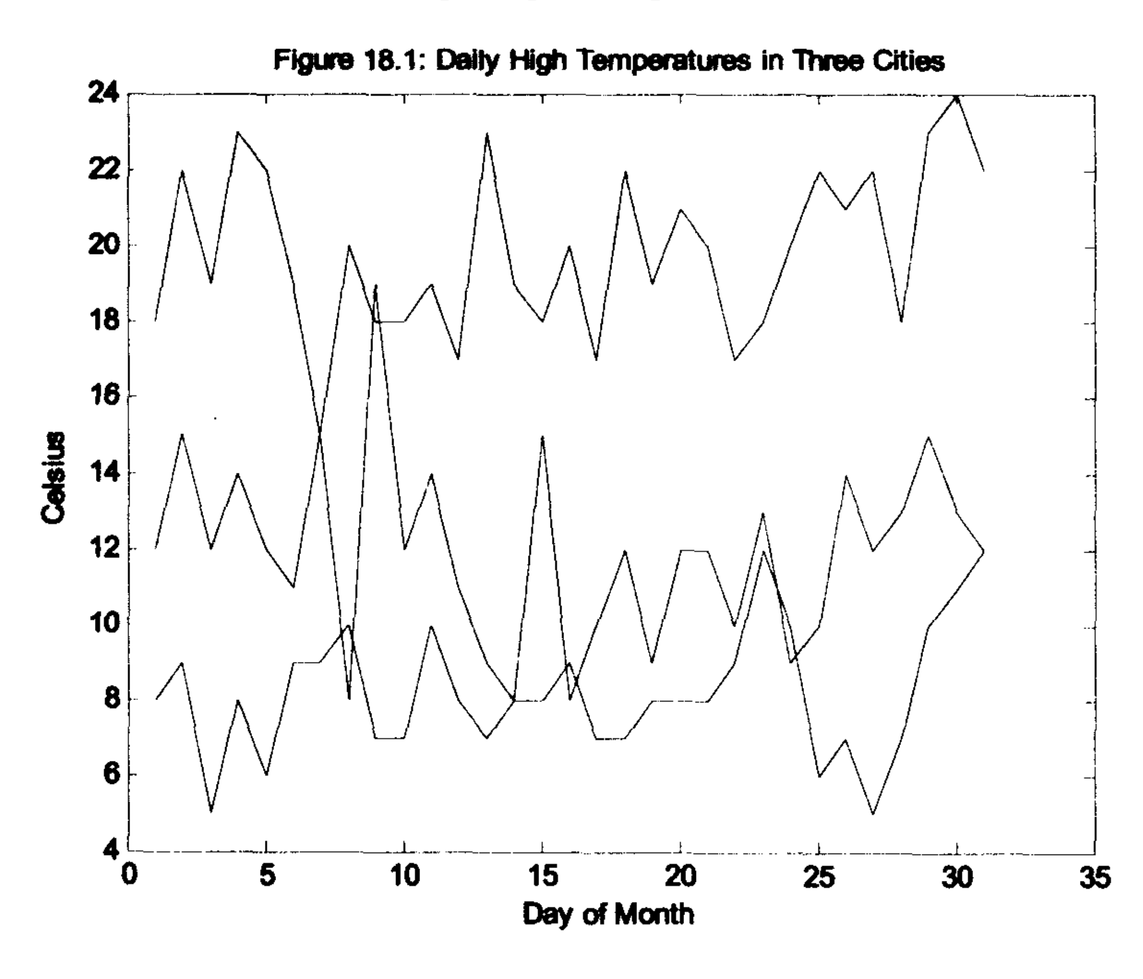


图 18.1 3 个城市的日最高气温

在上边的 plot 命令中，由于变量 d 是一个长度为 31 的向量，而 temps 是一个 31×3 的矩阵，因此，plot 命令就将 temps 的每一列以 d 为横坐标进行绘图。

我们就利用上面的数据，来看一下如何在 Matlab 中对数据进行分析。首先我们可以求出每个城市本月的月平均最高气温，代码如下：

```
>> avg_temp = mean(temps)
avg_temp =
    11.968  8.2258  19.871
```

上面的结果是 Matlab 分别求每一列的平均值得到的。由结果可知，第 3 个城市的月平均气温最高。如果我们对上述平均值再求平均值，如下面的代码所示：

```
>> avg_avg = mean(avg_temp)
avg_avg =
    13.3548
```

就得到了这 3 个城市的总体平均最高气温。

> 当给数据分析函数输入一个行向量或列向量时，Matlab 对向量进行数据分析后的结果将是一个标量。

Matlab 不仅针对列进行数据分析，还可以针对其他维进行数据分析。例如，下面的代码分别对 temps 的每列和每行取平均值：

```
>> avg_temp = mean(temps,1)  % same as above, work down the rows
avg_temp =
         11.968       8.2258       19.871
>> avg_tempr = mean(temps,2)  % compute means across columns
avg_tempr =
          12.667
          15.333
              12
              15
          13.333
              13
              13
          12.667
          14.667
          12.333
          14.333
              12
              13
          11.667
          13.667
          12.333
          11.333
          13.667
              12
          13.667
          13.333
              12
          14.333
              13
          12.667
```

```
            14
            13
        12.667
            16
            16
        15.333
```

由上可知，要想使 **mean** 函数对其他的维进行操作，需要为其提供第二个输入参数来指定进行操作的维。其中，1 代表对列求均值（默认）；2 代表对行求均值。

mean 以及其他统计函数也可以对多维数组进行操作，请看下面的例子：

```
>> temps2 = temps+round(2*rand(size(temps))-1);
>> temps3 = cat(3,temps,temps2);
>> size(temps3)
ans  =
     31    3     2
>> mean(temps3)   % same as mean(temps3,1)
ans(:,:,1)  =
         11.968        8.2258        19.871
ans(:,:,2)  =
         11.677        8.0968        19.903

>> squeeze(mean(temps3))' % squeeze to two dimensions
ans =
         11.968        8.2258        19.871
         11.677        8.0968        19.903

>> reshape(mean(temps3),3,2)' % alternate squeeze
ans =
         11,968        8.2258        19.871
         11.677        8.0968        19.903
>> mean(temps3,3)
ans =
       12          8         18
     14.5          9       22.5
       12        5.5         19
       14        7.5       23.5
     11.5        6.5       21.5
     10.5        9.5       18.5
       15        8.5       15.5
        8        9.5         20
     18.5        6.5       18.5
     11.5        6.5         18
     13.5         10         19
       11        7.5         17
      8.5        7.5       22.5
        8        7.5       18.5
       15        7.5       17.5
        8        9.5         20
       10          7       17.5
     11.5          7         22
        9          8       19.5
```

```
    12         8        21
    12         8        20
   9.5         9        17
    13      11.5        18
     9       9.5      20.5
    10         6      21.5
    14       7.5        21
    12       5.5      21.5
  12.5         7        18
    15       9.5        23
    13      11.5      24.5
  12.5      11.5        22
```

在上面的代码中，temps2 包含了 3 个城市第二个月（仍为 31 天）随机生成的最高气温数据。变量 temps3 则是一个三维数组，其第一页包含了第一个月的气温数据，第二页包含了第二个月的气温数据。命令 mean(temps3)对每一页的各列求平均值，结果得到一个 1 行、3 列、2 页的数组。由于上述结果中含有一个单一维，因此代码使用了函数 squeeze 或 reshape 将单一维去掉，使结果变成一个两行三列的二维数组。命令 mean(temps3,3)则在页的方向上求平均值，其结果是一个 31 行、3 列的二维数组，代表某个指定城市在所研究的两个月份中，指定某一天的温度平均值。

我们仅得到平均气温值还不够，还需要知道每个城市每天的最高气温与平均气温之间的温差。有的读者可能想当然地用下面的代码进行计算：

```
>> avg_temp
avg_temp =
    11.968  8.2258  19.871

>> temps - avg_temp
??? Error using ==> -
Matrix dimensions must agree.
```

上面的代码之所以行不通，是因为我们不能强制将两个不同维数的数组进行相减操作（temps 是 31×3 的数组，而 avg_temp 是一个 1×3 的数组）。由于两个变量的列数相等，利用 For 循环分别对每一行进行减操作也许是最直观简单的方法，如下面的代码所示：

```
>> for c =1:3
        tdev(:,c) = temps(:,c) - avg_temp(c);
   end
```

上述方法虽然可行，但是比较耗时。我们可以采用另一种不太直观但效率相对较高的方法，即先复制 avg_temp 使其维数等于 temps 的维数，然后再进行减法操作，如下面的代码所示：

```
>> tdev = temps - avg_temp(ones(31,1),:)
tdev =
  0.0323     -0.2258     -1.8710
  3.0323      0.7742      2.1290
  0.0323     -3.2258     -0.8710
  2.0323     -0.2258      3.1290
```

```
   0.0323   -2.2258    2.1290
  -0.9677    0.7742   -0.8710
   3.0323    0.7742   -4.8710
  -3.9677    1.7742    0.1290
   7.0323   -1.2258   -1.8710
   0.0323   -1.2258   -1.8710
   2.0323    1.7742   -0.8710
  -0.9677   -0.2258   -2.8710
  -2.9677   -1.2258    3.1290
  -3.9677   -0.2258   -0.8710
   3.0323   -0.2258   -1.8710
  -3.9677    0.7742    0.1290
  -1.9677   -1.2258   -2.8710
   0.0323   -1.2258    2.1290
  -2.9677   -0.2258   -0.8710
   0.0323   -0.2258    1.1290
   0.0323   -0.2258    0.1290
  -1.9677    0.7742   -2.8710
   1.0323    3.7742   -1.8710
  -2.9677    1.7742    0.1290
   0.0323   -3.2258    2.1290
   1.0323   -1.2258   -1.8710
   3.0323    1.7742    3.1290
   1.0323    2.7742    4.1290
   0.0323    3.7742    2.1290
```

命令 avg_temp(ones(31,1),:)将 avg_temp 的第一行（也是惟一的一行）复制 31 次，生成一个 31×3 的矩阵，其第 i 列的所有元素都等于 avg_temp(i)。另外，如果将 avg_temp(ones(31,1),:)用等价的命令 repmat(avg_temp,31,1)代替，也许读者看起来会直观一些。

Matlab 还可以求一组数据的最大值和最小值。例如，下面的代码求每个城市本月的最高气温的最大值：

```
>> max_temp = max(temps)
max_temp =
    19  12  24
```

可见 max 函数也是针对一列数据进行计算的。下面的代码同时还返回了气温最高的日期：

```
>> [max_temp,maxday] = max(temps)
max_temp =
    19  12  24
maxday =
    9   23  30
```

min 函数与 max 函数用法一样，只不过它用来求数组的最小值。例如，下面的代码求每个城市本月的最高气温的最小值：

```
>> min_temp = min(temps)
```

```
min_temp =
    8    5    15
```

下面的代码同时还返回了最高气温的最小值的日期：

```
>> [min_temp,minday] = min(temps)
min_temp =
    8    5    15
minday =
    8    3    7
```

除了上面介绍的均值、最大值和最小值外，Matlab 还提供一些其他的标准数据统计函数。我们先看下面的例子：

```
>> s_dev = std(temps) % standard deviation in each city
s_dev =
        2.5098      1.7646      2.2322
>> median(temps) % median temperature in each city
ans =
        12          8           20
>> cov(temps) % covariance
ans =
          6.2989     0.04086     -0.13763
         0.04086       3.114     0.063441
        -0.13763    0.063441       4.9828
>> corrcoef(temps) % correlation coefficients
ans =
         1          0.0092259   -0.024567
         0.0092259          1    0.016106
        -0.024567   0.016106            1
```

上面的代码分别求 3 个城市一个月内日平均最高气温的标准偏差、中值、方差和相关系数。

另外，用户还可以利用函数 diff 计算出相邻两天最高气温之间的差别：

```
>> daily_change = diff(temps)
daily_change =
         3        1        4
        -3       -4       -3
         2        3        4
        -2       -2       -1
        -1        3       -3
         4        0       -4
        -7        1        5
        11       -3       -2
        -7        0        0
         2        3        1
        -3       -2       -2
        -2       -1        6
        -1        1       -4
         7        0       -1
        -7        1        2
```

```
         2        -2        -3
         2         0         5
        -3         1        -3
         3         0         2
         0         0        -1
        -2         1        -3
         3         3         1
        -4        -2         2
         1        -4         2
         4         1        -1
        -2        -2         1
         1         2        -4
         2         3         5
        -2         1         1
        -1         1        -2
```

上述结果实际上描述了日最高气温的波动情况。例如，daily_change 的第一行是在这个月的第一天和第二天之间的日最高气温变化量。与 mean 函数一样，diff 还可以对其他维进行计算，例如，下面的代码计算了日最高气温在城市之间的差异（即在行方向的差值）：

```
>> city_change = diff(temps,1,2)
city_change =
        -4        10
        -6        13
        -7        14
        -6        15
        -6        16
        -2        10
        -6         6
         2        10
       -12        11
        -5        11
        -4         9
        -3         9
        -2        16
         0        11
        -7        10
         1        11
        -3        10
        -5        15
        -1        11
        -4        13
        -4        12
        -1         8
        -1         6
         1        10
        -4        16
        -7        14
        -7        17
        -6        11
        -5        13
```

```
    -2      13
     0      10
```

上述结果的第一列是第二个城市和第一个城市之间的差值；第二列是第三个城市和第二个城市之间的差值。

由于大多数函数对 NaNs 的操作都返回 NaNs，因此，我们可以利用 NaNs 来对数据进行查漏补缺。我们仍使用前面的数据，首先在数据中添加一些 NaNs：

```
>> temps4 = temps;      % copy data
>> temps4(5,1) = nan;   % insert some NaNs
>> temps4(29,2) = nan;
>> temps4(13:14,3) = nan
temps4 =
         12         8        18
         15         9        22
         12         5        19
         14         8        23
        NaN         6        22
         11         9        19
         15         9        15
          8        10        20
         19         7        18
         12         7        18
         14        10        19
         11         8        17
          9         7       NaN
          8         8       NaN
         15         8        18
          8         9        20
         10         7        17
         12         7        22
          9         8        19
         12         8        21
         12         8        20
         10         9        17
         13        12        18
          9        10        20
         10         6        22
         14         7        21
         12         5        22
         13         7        18
         15       NaN        23
         13        11        24
         12        12        22
```

然后，我们对 temps4 执行下面的统计操作：

```
>> max(temps4) % max and min ignore NaNs
ans=
     19     12     24
>> mean(temps4) % other statistical functions propagate NaNs
```

```
ans =
   NaN   NaN   NaN
>> std(temps4)
ans  =
   NaN   NaN   NaN
```

我们发现，除了 max 函数外，其他操作都得出了 NaNs 的结果，表明数据中有些数据丢失了（这些丢失的数据用 NaNs 进行了填充）。

如果用户要忽略丢失的数据，仍对现有的数据进行统计计算，就需要自己编程来跳过 NaNs 元素。例如，下面的例子利用 isnan 函数使 mean 函数在计算时跳过 NaNs：

```
>> m = zeros(1,3); % preallocate memory for faster results

>> for i=1:3        % find mean column by column
       idx = ~isnan(temps4(:,1));
       m(i)=mean(temps4(idx,i));
    end
>> m
m =
             13       7.3333      19.667
```

上例中，isnan 函数是一个逻辑函数，它检查一个数组中是否含有 NaNs，并在 NaNs 元素的位置返回 True。由于各列的 NaNs 元素的个数可能不同，因此无法使用一个命令同时排除所有列中的 NaNs 元素，必须使用 for 循环逐列排出。

另一种排除 NaNs 的方法是将数组中的 NaNs 用 0 替换，如下面的代码所示：

```
>> lnan = isnan(temps4);    % logical array identifying NaNs
>> temps4(lnan) = 0;        % change all NaNs to zero
>> n = sum(~lnan);          % number of nonNaN elements per column
>> m = sum(temps4)./n       % find mean for all columns
m =
        13      7.3333      19.667
```

上例中，首先将数组 temps4 中的 NaNs 置换成 0，然后利用 sum 函数求各列的平均值。

18.2 基本数据分析

除了统计数据分析之外，Matlab 还提供了多个通用数据分析函数。例如，我们可以利用函数 filter 对上节的温度数据进行滤波，如下面的代码所示：

```
>> filter(ones(1,4),4,temps)
ans=
         3          2         4.5
      6.75       4.25          10
      9.75        5.5       14.75
     13.25        7.5        20.5
     13.25          7        21.5
     12.25          7       20.75
        13          8       19.75
      11.5        8.5          19
```

```
   13.25      8.75        18
    13.5      8.25     17.75
   13.25       8.5     18.75
      14         8        18
    11.5         8     19.25
    10.5      8.25      19.5
   10.75      7.75     19.25
      10         8        20
   10.25         8      18.5
   11.25      7.75     19.25
    9.75      7.75      19.5
   10.75       7.5     19.75
   11.25      7.75      20.5
   10.75      8.25     19.25
   11.75      9.25        19
      11      9.75     18.75
    10.5      9.25     19.25
    11.5      8.75     20.25
   11.25         7     21.25
   12.25      6.25     20.75
    13.5      7.25        21
   13.25      8.25     21.75
   13.25        10     21.75
```

命令 filter(ones(1,4) ,4,temps)将利用下面的滤波器对 temps 进行过滤：$4y_n=x_n+x_{n-1}+x_{n-2}+x_{n-3}$ 或者 $y_n=(x_n+x_{n-1}+x_{n-2}+x_{n-3})/4$。也就是说，temps 的每一列都通过一个长度为 4 的移动平均滤波器进行滤波。通过给输入输出系数向量指定不同的值，可以形成各种各样的滤波器结构。

函数调用 y=filter(b,a,x)将采用下面的抽头延迟线算法对数据 x 进行滤波：

$$\sum_{k=0}^{N} a_{k+1} y_{n-k} = \sum_{k=0}^{M} b_{k+1} x_{n-k}$$

其中，$a=\{a_1,a_2,\cdots,a_{N+1}\}$ 是输出抽头权向量，$b=\{b_1,b_2,\cdots,b_{M+1}\}$ 是输入抽头权向量。当 N=2，M=3 时，则上边的等式可扩展为：

$$a_1 y_n + a_2 y_{n-1} + a_3 y_{n-2} = b_1 x_n + b_2 x_{n-1} + b_3 x_{n-2} + b_4 x_{n-3}$$

实际上，filter 函数内部使用的是滤波器的差分方程描述方式，该方式可以从前面的等式得到，即：

$$y_n = -\frac{1}{a_1}\sum_{k=1}^{N} a_{k+1} y_{n-k} + \frac{1}{a_1}\sum_{k=0}^{M} b_{k+1} x_{n-k}$$

当滤波器以状态空间的描述方式给出时，可以使用 Matlab 的内置函数 ltitr 对数据进行滤波。我们首先通过 ltitr 的帮助文本看一下该函数的使用方法：

```
LTITR   Linear time-invariant time response kernel.

X = LTITR(A,B,U) calculates the time response of the system:
        x[n+1] = Ax[n] + Bu[n]
to input sequence U. The matrix U must have as many columns as
there are inputs u. Each row of U corresponds to a new time
point. LTITR returns a matrix X with as many columns as the
number of states x, and with LENGTH(U) rows.
    LTITR(A,B,U,Xo) can be used if initial conditions Xo exist.

Here is what it implements, in high speed:
    for i=1:n
        x(:,i) = xo;
        xo = a*xo+b*u(i,:).';
    end
    x = x.';
```

如果滤波器的状态空间输出方程为：$y[n]=Cx[n]+Du[n]$，其中 C 和 D 分别为指定维数的矩阵，则可使用命令 x=ltitr(A,B,u)求出滤波器的状态响应 x[n]，然后再利用下面的命令求出滤波器的输出 y：

```
y = x*C.' +u*D.';
```

其中 y 的列数等于输出的个数，y 的行数等于时间样点数。

基本数据分析的另一项重要任务是对数据进行排序，在 Matlab 中，这一工作通常采用 sort 函数实现。例如，下面的代码对一个随机生成的列向量按从小到大的顺序重新排列，并返回排序后的索引向量：

```
>> data = rand(10,1) % create some data
data =
        0.61543
        0.79194
        0.92181
        0.73821
        0.17627
        0.40571
        0.93547
        0.9169
        0.41027
        0.89365
>> [sdata,sidx] = sort(data) % sort in ascending order
sdata =
        0.17627
        0.40571
        0.41027
        0.61543
        0.73821
        0.79194
        0.89365
        0.9169
```

```
         0.92181
         0.93547
sidx =
         5
         6
         9
         1
         4
         2
        10
         8
         3
         7
```

有时候用户需要知道一个数据在整个向量中的位次。例如，用户若想知道未排序的数组中的第 i 个数据在排序后的数组中的位次是第几个，可以使用下面的命令找到答案：

```
>> ridx(sidx) = 1:10        % ridx is rank
ridx =
     4    6    9    5    1    2   10   8    3    7
```

从结果可以看出，未排序的数组中的第一个元素出现在排序后的数组的第 5 个元素的位置，而排序后的最后一个元素（第 10 个元素）则是排序前的第 7 个元素。

在默认情况下，sort 函数总以升序对数组进行排序。如果用户需要降序排序，则可使用下面的函数调用方式：[sdata,sidx]=sort(data,'descend')。在该调用方式中，'descend'表示按降序排序，sdata 返回排序后的数组，sidx 返回排序后的索引。

当被排序的数组是一个矩阵（如前面例子中的 temps）时，则 sort 将对矩阵的每一列进行排序，并在每一列生成一个排序后的索引。与向量排序一样，在默认情况下，sort 对矩阵的每列进行排序，要想使其对矩阵的每行进行排序，就需要给 sort 添加一个输入参数'descend'。另外，要想按其他方向进行排序，还需要在'descend'参数前面再添加一个输入参数，表明要对那个方向的数据进行排序。例如，要对矩阵 temps 的行按降序进行排序时，可采用下面的函数调用方式：[sdata,sidx]=sort(temps,2,'descend')。

在很多情况下，用户希望执行类似于电子工作表格中的排序，即只对一个数组的某一列进行排序，然后将这个排序后的顺序应用到剩下的所有列上。下面的代码就给出了一个简单实例：

```
>> data = randn(10,4) % new data for sorting
data =
      -0.43256       -0.18671        0.29441       -0.39989
       -1.6656        0.72579        -1.3362           0.69
       0.12533       -0.58832        0.71432        0.81562
       0.28768         2.1832         1.6236        0.71191
       -1.1465        -0.1364       -0.69178         1.2902
        1.1909        0.11393          0.858         0.6686
        1.1892         1.0668          1.254         1.1908
     -0.037633       0.059281        -1.5937        -1.2025
       0.32729      -0.095648         -1.441       -0.01979
       0.17464       -0.83235        0.57115       -0.15672
```

```
>> [tmp,idx] = sort(data(:,2)); % sort second column
>> datas = data(idx,:) % shuffle rows using idx from 2nd column
datas =
         0.17464        -0.83235        0.57115       -0.15672
         0.12533        -0.58832        0.71432        0.81562
        -0.43256        -0.18671        0.29441       -0.39989
         -1.1465         -0.1364       -0.69178         1.2902
         0.32729       -0.095648         -1.441       -0.01979
       -0.037633        0.059281        -1.5937        -1.2025
          1.1909         0.11393          0.858         0.6686
         -1.6656         0.72579        -1.3362           0.69
          1.1892          1.0668          1.254         1.1908
         0.28768          2.1832         1.6236        0.71191
```

上面代码中，只对随机数组的第二列按照升序进行排序，然后利用排序后的索引对其他列进行重排。也就是说，data 中第二列元素的大小顺序决定了整个数组各行的排序，例如，由于 data 的第二列的最后一个元素（-0.83235）是该行最小的元素，因此，排序后导致 data 的最后一行全部移到 datas 的第一行。

对一个向量进行排序其实就是使该向量满足严格单调性。所谓单调性指一个向量的元素总是沿着该向量的方向增大或减小。在 Matlab 中，可以使用函数 diff 判断一个向量是否满足单调性，如果一个向量经过 diff 后是非正值或非负值，则说明该向量是单调的，否则说明该向量不是单调的。例如，下面的代码分别判断一个随机向量和该向量经排序后的单调性：

```
>> A = diff(data) % check random data
A =
          0.1765
         0.12988
        -0.18361
        -0.56194
         0.22944
         0.52976
       -0.018565
        -0.50663
         0.48338
>> mono = all(A>0) | all(A<0) % as expected, not monotonic
mono =
      0
>> B = diff(sdata) % check random data after sorting
B =
        0.22944
        0.004564
        0.20516
        0.12277
        0.05373
        0.10171
        0.023255
        0.0049085
        0.013657
```

```
>> mono = all(B>0) | all(B<0) % as expected, monotonic
mono =
      1
```

另外，如果两次调用 diff 函数，还可以判断一个向量是否是等间隔单调。例如，下面的代码显示 sdata 不是等间隔单调的，而 1:25 这个向量是等间隔单调的：

```
>> all(diff(diff(sdata))==0)    % random data is not equally spaced
ans =
    0
>> all(diff(diff(1:25))==0)     % but numbers from 1 to 25 are equally spaced
ans =
    1
```

18.3 数据分析和统计函数

在 Matlab 中，数据分析和统计大都是针对矩阵进行操作的，矩阵中不同的列代表不同的变量，每行则代表了该变量不同的观测值。在默认情况下，Matlab 的数据分析和统计函数都是对矩阵中每列数据进行运算的，要想使它们沿着其他的方向进行运算，需要在函数调用时增加一个输入参数，指定函数运算的方向。由于 Matlab 的数据分析和统计函数都与前面介绍的一些函数在用法上大同小异，因此，本节只在下表中给出 Matlab 中常用的数据分析和统计函数的简单描述。

函数	描述
corrcoef(A)	求 A 的相关系数
cov(A)	求 A 的协方差矩阵
cplxpair(v)	将向量按复数共轭对重新排序
cumprod(A)	求 A 的元素累加积
cumsum(A)	求 A 的元素累加和
cumtrapz(A)	求 A 的累计梯形积分
del2(A)	求 A 的离散 Laplace 变换（表面曲率）
diff(A)	求 A 的相邻元素间差值
gradient(Z,dx,dy)	计算近似表面梯度
histc(X,edges)	使用 edges 标记的容器的直方图计数和容器位置
issorted(A)	判断 A 是否已被排序
max(A)	返回 A 的各列的最大值
median(A)	求 A 的各列的中值
mean(A)	求 A 的各列的平均值
min(A)	返回 A 的各列的最小值
prod(A)	求 A 的每列中各元素的积
sort(A)	对 A 的每列按升序或降序排序
sortrows(A)	对 A 的每行按升序排序，也称为字典排序

（续表）

函数	描述
std(A)	求 A 的标准偏差
sum(A)	求 A 的每列中各元素的和
trapz(A)	利用梯形法求数值积分
var(A)	求 A 的方差，也就是标准差的平方

Chapter 19

数 据 插 值

有时候，我们并不仅仅对给定数据点处的函数值感兴趣，还对这些数据点中间的某些点处的函数的值感兴趣。当用户无法快速地对这些中间点执行函数运算时（例如，这些中间数据需要进行专门的实验测量或者进行长时间的计算才能得到时），就需要用到数据插值的概念。Matlab 提供了对数组的任何一维进行插值的工具，这些工具大都需要用到多维数组的操作。限于篇幅，本章将对 1 维和 2 维插值进行详细说明，对多维数组的插值问题只进行了简单介绍。

19.1 一维插值

其实数据插值的概念并不难理解，在我们用 Matlab 进行绘图时，就用到了数据插值。由于 Matlab 在默认情况下需要用连续的直线来连接各个数据点，那么这些数据点之间的各个数据就需要通过线性插值的方法获得。线性插值是数据插值的一种形式，该方法假设两个数据点之间的中间值都落在这两个数据点连成的直线上。很明显，随着数据点数目增加和数据点之间的距离缩短，线性插值会变得越来越精确，Matlab 绘制的曲线也越精确。例如，下面的代码针对不同的数据点个数绘制了 sin 函数的图形：

```
>> x1 = linspace(0,2*pi,60);
>> x2 = linspace(0,2*pi,6);
>> plot(x1,sin(x1),x2,sin(x2),'--')
>> xlabel('x'),ylabel('sin(x)')
>> title('Figure 19.1: Linear Interpolation')
```

从图 19.1 可以看到，利用 60 个数据点绘制的图形要比利用 6 个数据点绘制的图形精确。

上面的Matlab绘图的例子可以称为一维插值的一个特例。为了更清楚地阐述一维插值，我们考虑一个更常见的例子。我们知道，人耳对声音的敏感度是随着声音频率的变化而变化的，下面的两个变量给出了科学家对人耳在不同声音频率下的灵敏度的测试数据（单位：分贝）。

```
>> Hz=[20:10:100 200:100:1000 1500 2000:1000:10000]; % frequencies in Hertz
>> spl =[76 66 59 54 49 46 43 40 38 22 ...          % sound pressure level in dB
```

```
14 9 6 3.5 2.5 1.4 0.7 0 -1 -3...
-8 -7 -2 2 7 9 11 12];
```

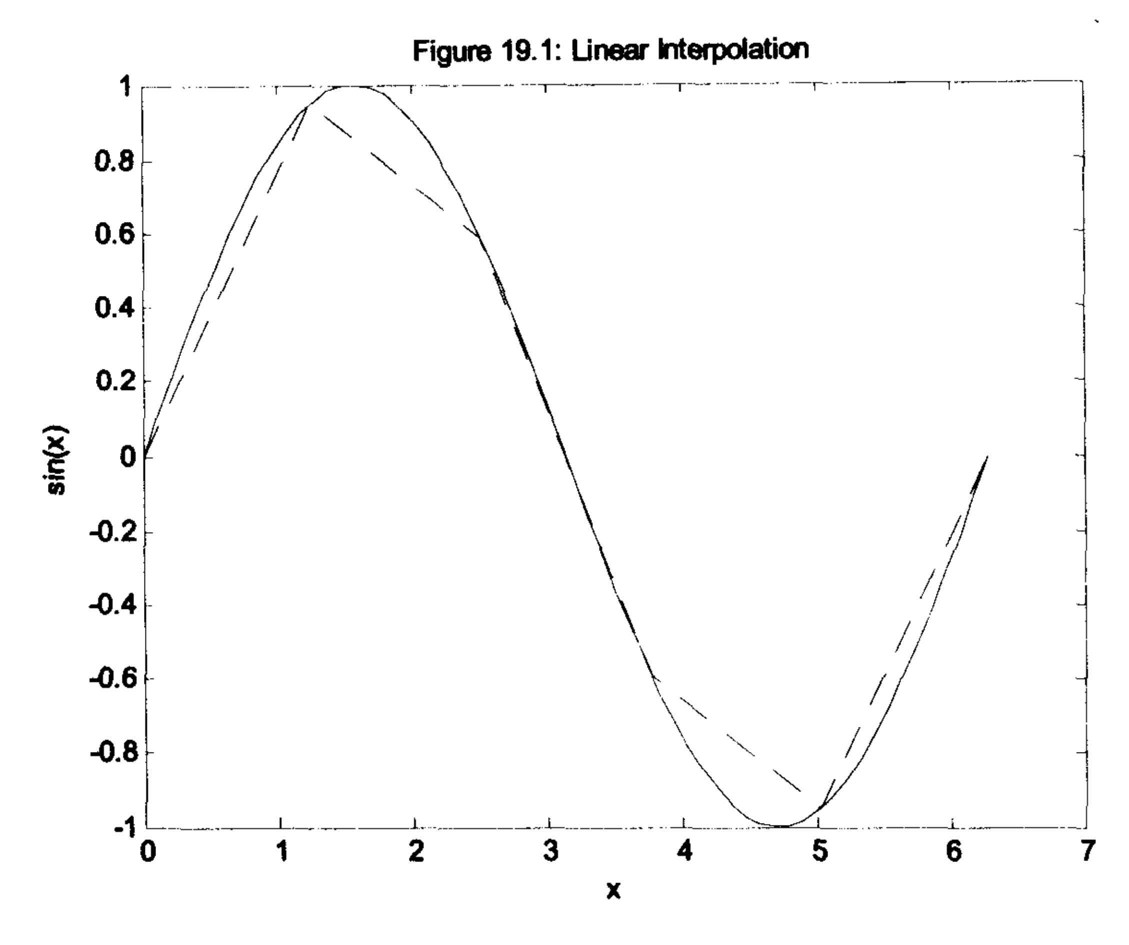


图 19.1 线性插值

上面的声音压力水平是归一化后的值，并以 1000Hz 处的值为 0 分贝。为了形象地看到人耳的这一变化规律，我们将上面的数据用图形绘制出来，由于频率范围跨度很大，因此我们在绘图时用到了对数 x 轴（即先对频率取对数，再画图），绘图代码如下所示：

```
>> semilogx(Hz,spl, '-o')
>> xlabel('Frequency,Hz')
>> ylabel('Relative Sound Pressure Level,dB')
>> title('Figure 19.2: Threshold of Human Hearing')
>> grid on
```

从图 19.2 可知，人耳对于大约 3kHz 的声音是最敏感的。

在 Matlab 中，数据的一维插值函数是 interp1。下面我们根据上述数据，利用此函数估计一下频率为 2.5kHz 时的声音压力水平。下面的代码给出了几个不同的插值估计方法：

```
>> s = interp1(Hz,spl,2.5e3)           % linear interpolation
s =
          -5.5
>> s = interp1(Hz,spl,2.5e3,'linear')  % linear interpolation again
s =
          -5.5
>> s = interp1(Hz,spl,2.5e3,'cubic')   % cubic interpolation
s =
      -6.0488
>> s = interp1(Hz,spl,2.5e3,'spline')  % cubic spline interpolation
s =
       -5.869
>> s = interp1(Hz,spl,2.5e3,'nearest') % nearest neighbor interpolation
```

```
s =
    -8
```

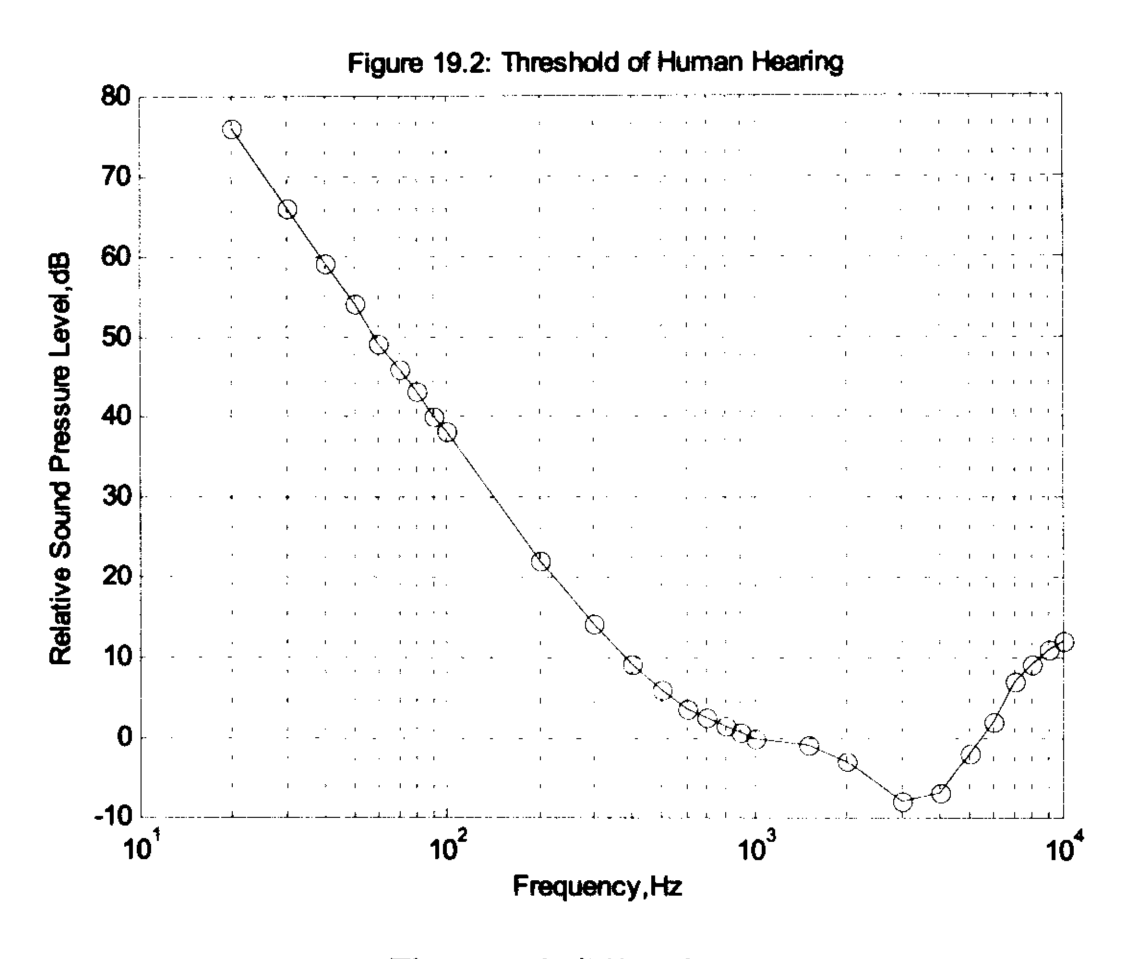


图 19.2　人类的听力门限

我们发现，采用不同的插值方法得到的结果是有差别的。前两个语句（实际上是一种方法，因为在默认情况下，interp1 函数使用线性插值）返回的结果最接近图 19.2 中的值，由此可见，这些数据之间存在的是一种近似线性的关系。立方（cubic）和样条（spline）插值通常用于满足三阶多项式的数据之间的插值，因此本例中这两种方法没有得出最近似的结果。本例中效果最差的插值是最邻近法（nearest）插值，因为该方法仅仅返回与插值点最邻近的一个原始输入数据的值。

那么，对一个给定的问题，用户该如何选择正确的插值方法呢？在大多数情况下，使用线性插值就可以满足数据处理的要求，因此，线性插值是 interp1 默认的插值方法。最相邻插值法虽然精确性最差，但在要求快速插值或者数据集合很大的情况下经常用到。立方插值和样条插值占用时间最多，但往往能得出最精确的结果。这些方法用户可根据不同的需要灵活选用。

上边的例子只考虑了在一个点上的插值，实际上，interp1 能够处理任意数量的数据点上的插值。当在两个相邻的数据点之间插入多个值时，就相当于画出这两个数据点之间的局部曲线，这时，采用立方插值或者样条插值法可以取得最平滑的效果。也就是说，当从更加精细的角度考察插值效果时，显然立方插值或者样条插值法是优于其他方法的。例如：

```
>> Hzi = linspace(2e3,5e3); % look closely near minimum
>> spli = interp1(Hz,spl,Hzi,'spline'); % interpolate near minimum
>> i = find(Hz>=2e3 & Hz<=5e3); % find original data indices near minimum

>> semilogx(Hz(i),spl(i),'--o',Hzi,spli) % plot old and new data
>> xlabel('Frequency,Hz')
>> ylabel('Relative Sound Pressure Level,dB')
```

```
>> title('Figure 19.3: Threshold of Human Hearing')
>> grid on
```

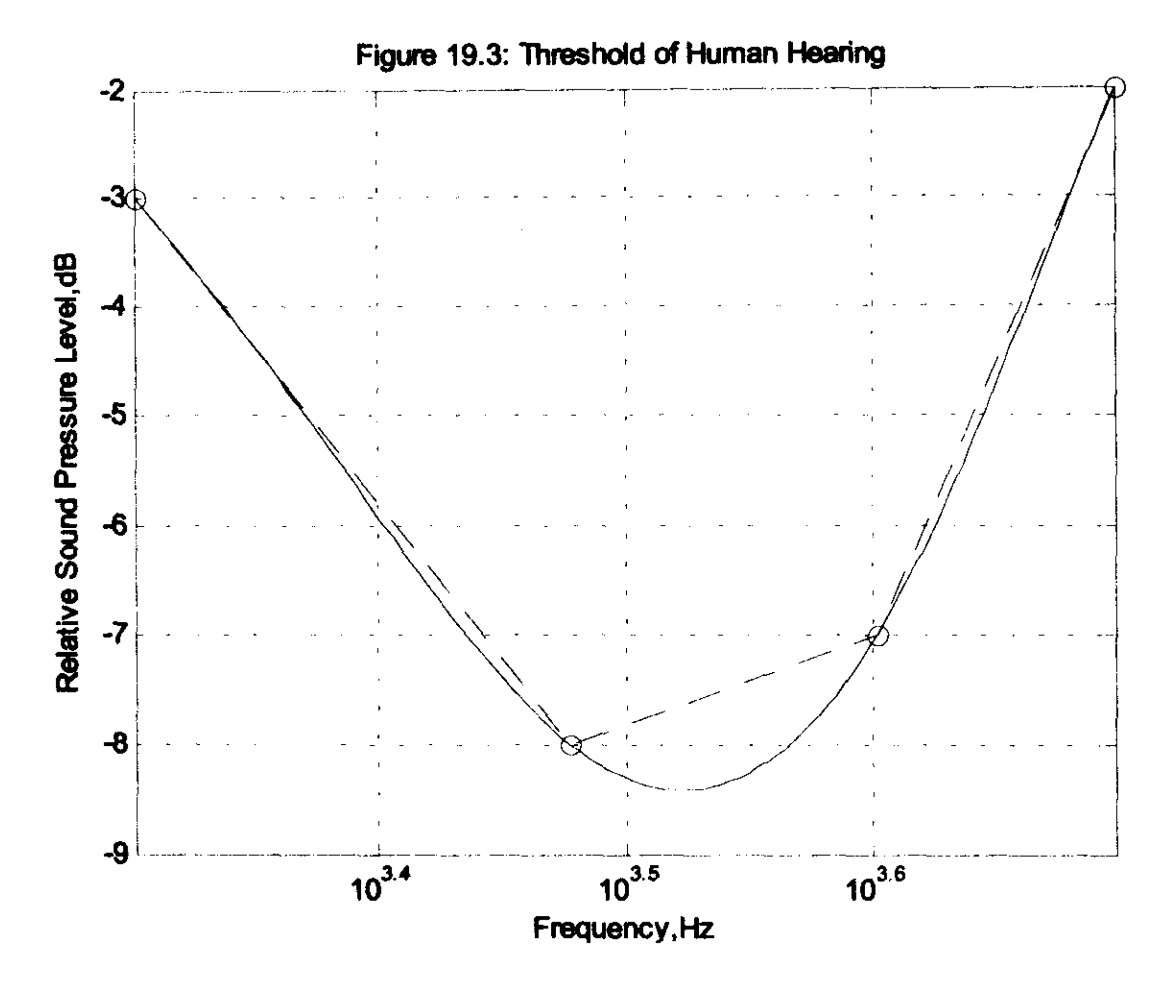


图 19.3 人耳的听觉门限

在图 19.3 中，虚线表示的是线性插值，实线表示的是立方插值，原始数据用圆圈标出。我们发现，当我们将频率轴放大来观察更细微的插值效果时，样条插值法显然比线性插值法获得的曲线更加平滑。尤其值得注意的是，样条插值法所得到的曲线不像线性插值法那样在数据点处存在斜率突变，其斜率变化是连续的。

利用上面的数据，我们可以对最大敏感度频率进行更准确的估计，代码如下：

```
>> [spl_min,i] = min(spli)      % minimum and index of minimum
spl_min =
      -8.4245
i =
    45
>> Hz_min = Hzi(i)              % frequency at minimum
Hz_min =
      3333.3
```

由此我们可断定，人耳对大约 3.33kHz 的声音频率最为敏感。

我们在使用 interp1 函数时应该注意到，该函数对其第一个输入变量（又叫独立变量）是有要求的。它要求该变量必须是单调的，也就是说，必须始终增大（单调递增）或者始终减小（单调递减）。例如，在前面的例子中，Hz 总是单调递增的。

最后，interp1 函数可以同时对多个数据集进行插值。也就是说，如果 x 是一个向量，不论 y 是一个向量（比如上面的 spl）还是一个有 length(x)个行，任意个列的数组，都可以同时对 y 的各列进行插值操作。例如，下面的代码同时对 3 个三角函数 sin(x)、cos(x)和 tan(x)进行插值：

```
>> x = linspace(0,2*pi,11)';    % example data
>> y = [sin(x) cos(x) tan(x)];
>> size(y) % three columns
ans =
    11      3
>> xi = linspace(0,2*pi); % interpolate on a finer scale
>> yi = interp1(x,y,xi,'cubic');
>> size(yi) % result is all three columns interpolated
ans =
    100     3
```

19.2 二维插值

前一节讲的一维插值主要对单变量函数进行插值，本节的二维插值则是对两个变量的函数（$z=f(x,y)$）进行插值。我们仍使用一个简单的例子来阐明二维插值的基本原理。假设一个海洋勘探公司要用声纳技术来绘制海底地形图。他们将测量的海域用 0.5 公里宽的方形格栅划分成不同的区域，并在栅格的每个交点处记录下测量的海洋深度（单位：米）以便日后分析。下面的 M 脚本文件 ocean.m 记录其中的一部分测量数据：

```
% ocean.m, example test data

% ocean depth data

x = 0:.5:4;  % x-axis (varies across the rows of z)
y = 0:.5:6;  % y-axis (varies down the columns of z)

z = [100    99    100    99    100    99    99    99    100
     100    99     99    99    100    99   100    99     99
      99    99     98    98    100    99   100   100    100
     100    98     97    97     99   100   100   100     99
     101   100     98    98    100   102   103   100    100
     102   103    101   100    102   106   104   101    100
      99   102    100   100    103   108   106   101     99
      97    99    100   100    102   105   103   101    100
     100   102    103   101    102   103   102   100     99
     100   102    103   102    101   101   100    99     99
     100   100    101   101    100   100   100    99     99
     100   100    100   100    100    99    99    99     99
     100   100    100    99     99   100    99   100     99];
```

为了便于分析，我们使用下面的代码将这些数据用三维图形绘制出来，如图 19.4 所示。

```
>> mesh(x,y,z)
>> xlabel('X-axis,km')
>> ylabel('Y-axis,km')
>> zlabel('Ocean Depth,m')
>> title('Figure 19.4: Ocean Depth Measurements')
```

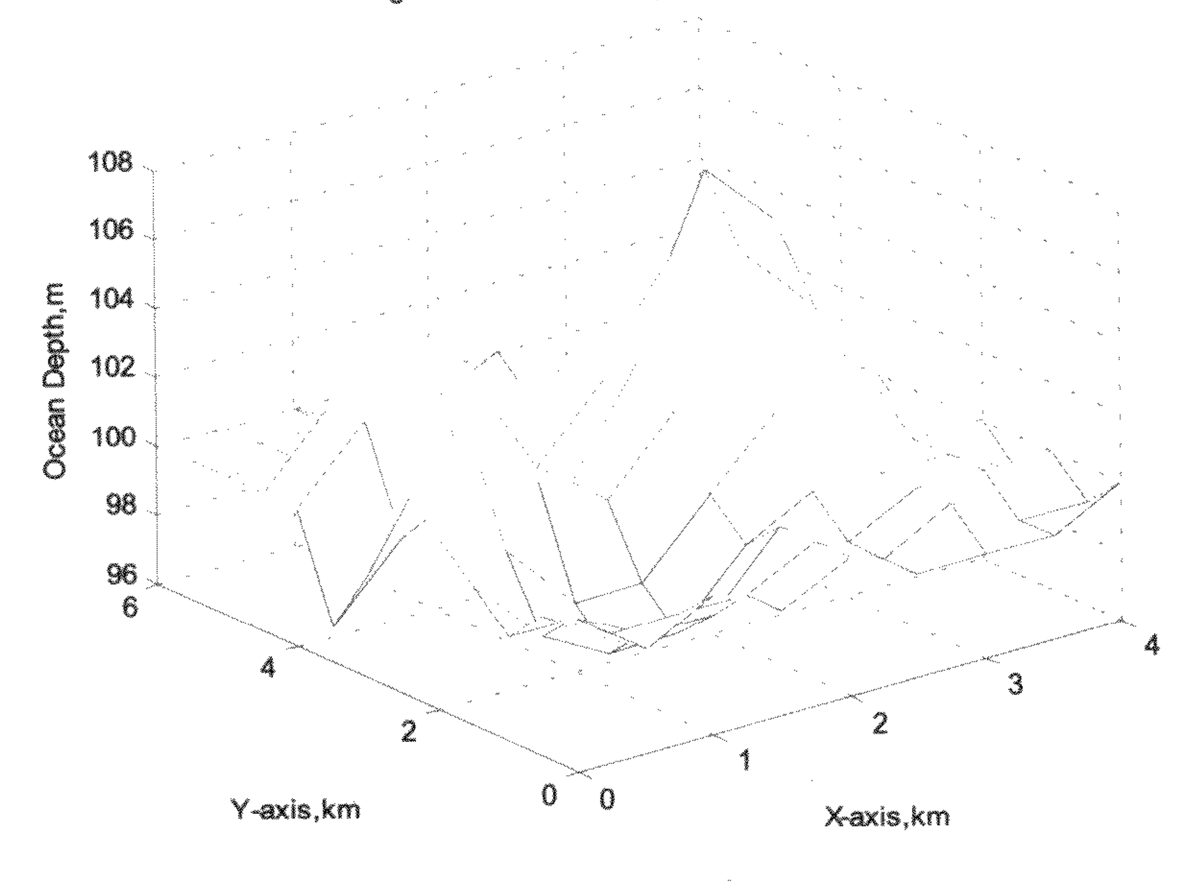


图 19.4 海洋深度测量

根据上面的数据，我们可以利用函数 interp2 测量栅格区域内任意一点的深度。例如，下面的代码测量(2.2,3.3)处的海洋深度：

```
>> zi = interp2(x,y,z,2.2,3.3)
zi =
        103.92
>> zi = interp2(x,y,z,2.2,3.3,'linear')
zi =
        103.92
>> zi = interp2(x,y,z,2.2,3.3,'cubic')
zi =
        104.19
>> zi = interp2(x,y,z,2.2,3.3,'nearest')
zi =
        102
```

在上面的代码中，Matlab 同样使用了几种不同的二维插值方法来测量(2.2,3.3)处的海洋深度，其中默认的方法也是线性插值法。

从图 19.4 可以看出，原始数据的三维图形是很不平滑的，为了利用数据插值使之平滑化，我们首先对 x 轴和 y 轴进行细化，如下面的代码所示：

```
>> xi = linspace(0,4,30);          % finer x-axis
>> yi = linspace(0,6,40);          % finer y-axis
```

细化后的测量区域将由更加密集的栅格构成。这时，我们希望对所有的栅格交点进行插值，也就是说，对所有的 xi 和 yi 的组合所构成的点进行插值。要获得所有的 xi 和 yi 的组合，需要使用 meshgrid 函数，该函数接受两个向量作为输入，生成这两个向量的元素间的所有可能的组合。我们先来看一个简单的例子了解一下 meshgrid 函数的用法：

```
>> xtest = 1:5
xtest =
      1    2    3    4    5

>> ytest = 6:9
ytest =
      6     7     8     9

>> [xx,yy] = meshgrid(xtest,ytest)
xx =
      1     2     3     4     5
      1     2     3     4     5
      1     2     3     4     5
      1     2     3     4     5
yy =
      6     6     6     6     6
      7     7     7     7     7
      8     8     8     8     8
      9     9     9     9     9
```

在上面的例子中，xx 有 length(ytest)个行，每行都由 xtest 构成；yy 有 length(xtest)个列，每列都由 ytest 构成。这样，我们将 xx 和 yy 对应的元素进行组合，就构成了 xtest 和 ytest 元素间的所有组合。

我们继续回到海洋深度测量的例子中，利用 meshgrid 函数，我们可以得到细化后的所有栅格交点的集合（即 xi 和 yi 所有元素的组合），具体实现代码如下：

```
>> [xxi,yyi] = meshgrid(xi,yi);     % grid of all combinations of xi and yi
>> size(xxi)                        % xxi has 40 rows each containing xi
ans =
    40  30
>> size(yyi)        % yyi has 30 columns each containing yi
ans =
    40  30
```

有了细化的栅格坐标 xxi 和 yyi，我们可以通过下面的代码，通过二维插值使海洋深度的三维图形变得更加平滑，平滑后的图形如图 19.5 所示。

```
>> zzi = interp2(x,y,z,xxi,yyi,'cubic');       % interpolate
>> size(zzi) % zzi is the same size as xxi and yyi
ans =
    40      30
>> mesh(xxi,yyi,zzi) % plot smoothed data
>> hold on
>> [xx,yy] = meshgrid(x,y);% grid original data
>> plot3(xx,yy,z+0.1,'ok') % plot original data up a bit to show nodes
>> hold off

>> xlabel('X-axis,km')
>> ylabel('Y-axis,km')
>> zlabel('Ocean Depth,m')
>> title('Figure 19.5: 2-D Smoothing')
```

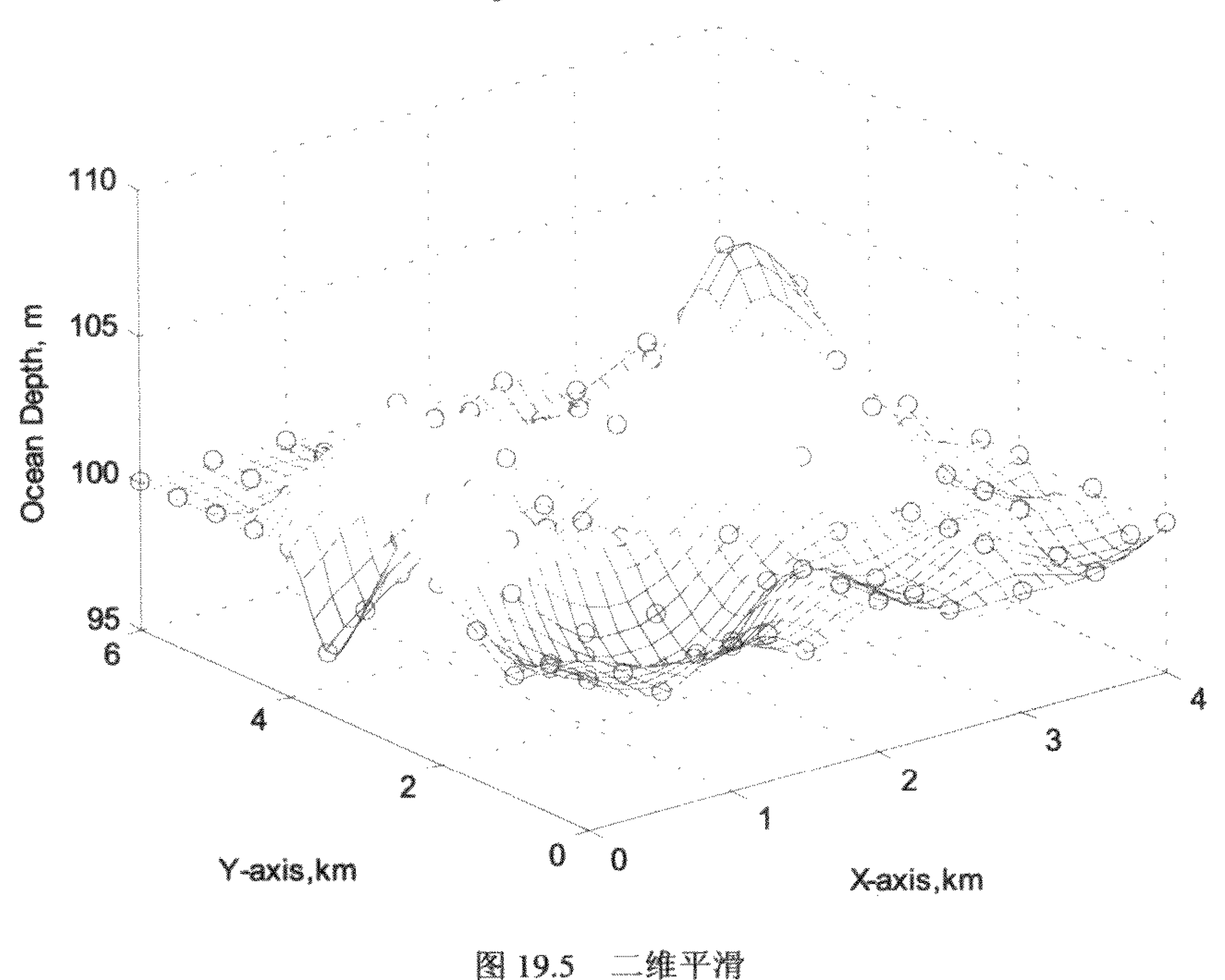


图 19.5 二维平滑

利用上面的插值数据，我们还可以通过下面的代码估计海洋深度的峰值及其位置：

```
>> zmax = max(max(zzi))
zmax =
        108.05
>> [i,j] = find(zmax==zzi);
>> xmax = xi(j)
xmax =
        2.6207
>> ymax = yi(i)
ymax =
        2.9231
```

根据前面介绍的一维和二维插值，用户能将插值技术很自然地扩展到更高的维数，Matlab 也提供了函数 ndgrid、interp3 和 interpn 来支持更高维数的插值运算。其中，ngrid 是多维网格构造函数，是函数 meshgrid 在多维情况下的扩展；interp3 用于完成三维空间的插值；interpn 用于完成更高维空间的插值。和 interp1 和 interp2 一样，interp3 和 interpn 也都提供了'linear'、'cubic'和'nearest'的插值方法。对 Matlab 而言，多维插值将使用多维数组来组织数据和进行插值操作。关于这些函数的更多详细信息请读者参看 Matlab 文档及在线帮助信息。

19.3 三角测量和分散数据

在有些应用场合（比如几何分析场合）中，待测量的数据点通常不像前两节所讨论的

例子中那样出现在一个集中的区域，而是分散分布的。例如，下面的代码给出了一个分散数据的实例：

```
>> x = randn(1,12);
>> y = randn(1,12);
>> z = zeros(1,12); % no z component for now
>> plot(x,y,'o')
>> title('Figure 19.6: Random Data')
```

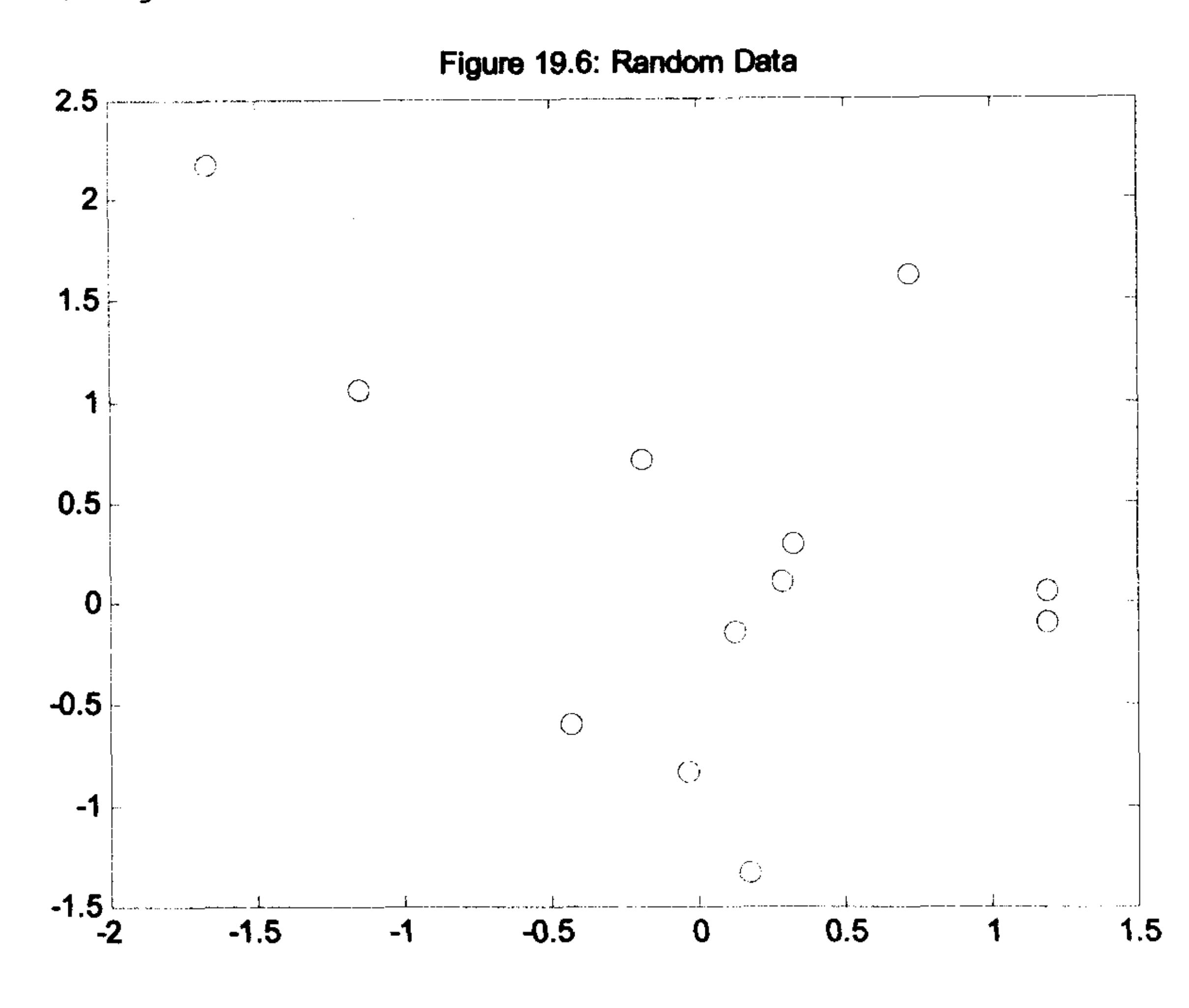


图 19.6 随机分散数据

在对上述分散数据进行插值之前，通常先使用 Delaunay 三角测量方法来分析数据，该方法用一组三角形将所有的数据点连接起来，并且没有任何一个数据点落在任何一个三角形之内。在 Matlab 中，函数 delaunay 用于完成上述三角测量，该函数接受分散的数据点并返回一列数据索引用于标明各个三角形的顶点。例如，我们对上面的分散数据使用 delaunay 函数，将返回如下的结果：

```
>> tri = delaunay(x,y)
tri =
     2     5     6
    10     6     5
     1     5     2
    12     1     2
     3     6    10
     1    10     5
    11     7    10
     7     3    10
     1    11    10
     8     7    11
```

```
     1      8     11
    12      8      1
     9      8     12
     4     12      2
     4      9     12
     9      7      8
     9      3      7
```

其中，每一行由三角形的 3 个顶点的 x 和 y 的索引构成。例如，第一个三角形是以位于 x([2 5 6])和 y([2 5 6])的 3 个数据点为顶点构成的。我们可以使用函数 trimesh 将这些三角形绘制出来：

```
>> hold on, trimesh(tri,x,y,z),hold off
>> hidden off
>> title('Figure 19.7: Delaunay Triangulation')
```

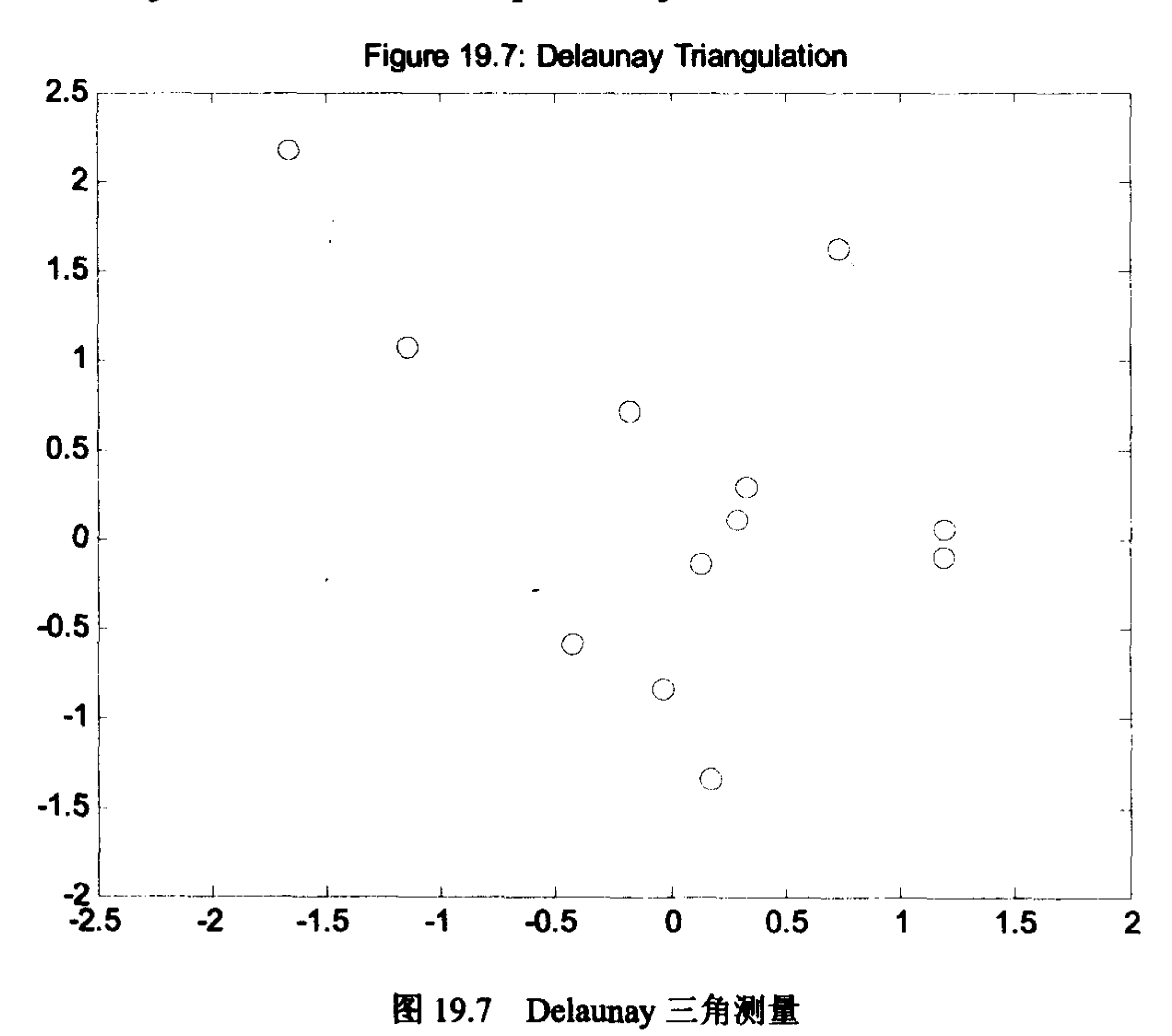


图 19.7 Delaunay 三角测量

获得 Delaunay 三角形测量数据 tri 之后，用户可以用函数 tsearch 和 dsearch 对分散数据进行插值。例如，可以使用 tsearch 函数查看包围原点的三角形是哪个三角形，代码如下：

```
>> tsearch(x,y,tri,0,0) % find row of tri closest to (0,0)
ans =
    13
>> tri(ans,:)   % vertices of triangle closest to (0,0)
ans =
    9    8    12
```

当然，tsearch 也可以同时接受多个输入值，查看这些点都由哪些三角形包围。例如，下面的代码查看点(-0.5,1)和(1,0.5)分别位于哪个三角形中：

```
>> tsearch(x,y,tri,[-.5 1],[1 .5])
ans =
    15   NaN
```

从结果可以看出，点(-0.5,0.1)位于第 15 个三角形中，而点(1,0.5)没有被任何一个三角形包围。

函数 dsearch 的用法与 tsearch 基本相同，只不过它返回最接近于被考察点的 x 和 y 的索引，而不是一个三角形。例如，下面的代码将返回与点(-0.5,1)和(1,0.5)最接近的数据点：

```
>> dsearch(x,y,tri,[-.5 1],[1 .5])
ans =
    12   7
```

从结果可以看出，与点(-0.5,1)最接近的点是(x(12),y(12))，与点(0.1,0.5)最接近的点是(x(7),y(7)) 。

由于分散数据的杂散性，有时我们有必要知道一组分散数据的边界，即这组分散数据的最外层数据连成的凸多边形。函数 convhull 将返回描述该多边形的 x 和 y 的索引，例如，下面的命令返回上述分散数据的边界多边形：

```
>> k = convhull(x,y)
k =
    2    6    3    9    4    2
```

注意，由于 k 的第一个和最后一个索引值相等，因此 convhull 返回的是一个封闭的凸多边形。根据 k 的值，可以画出上述分散数据集的凸多边形边界，代码如下：

```
>> plot(x,y,'o',x(k),y(k))
>> title('Figure 19.8: Convex Hull ')
```

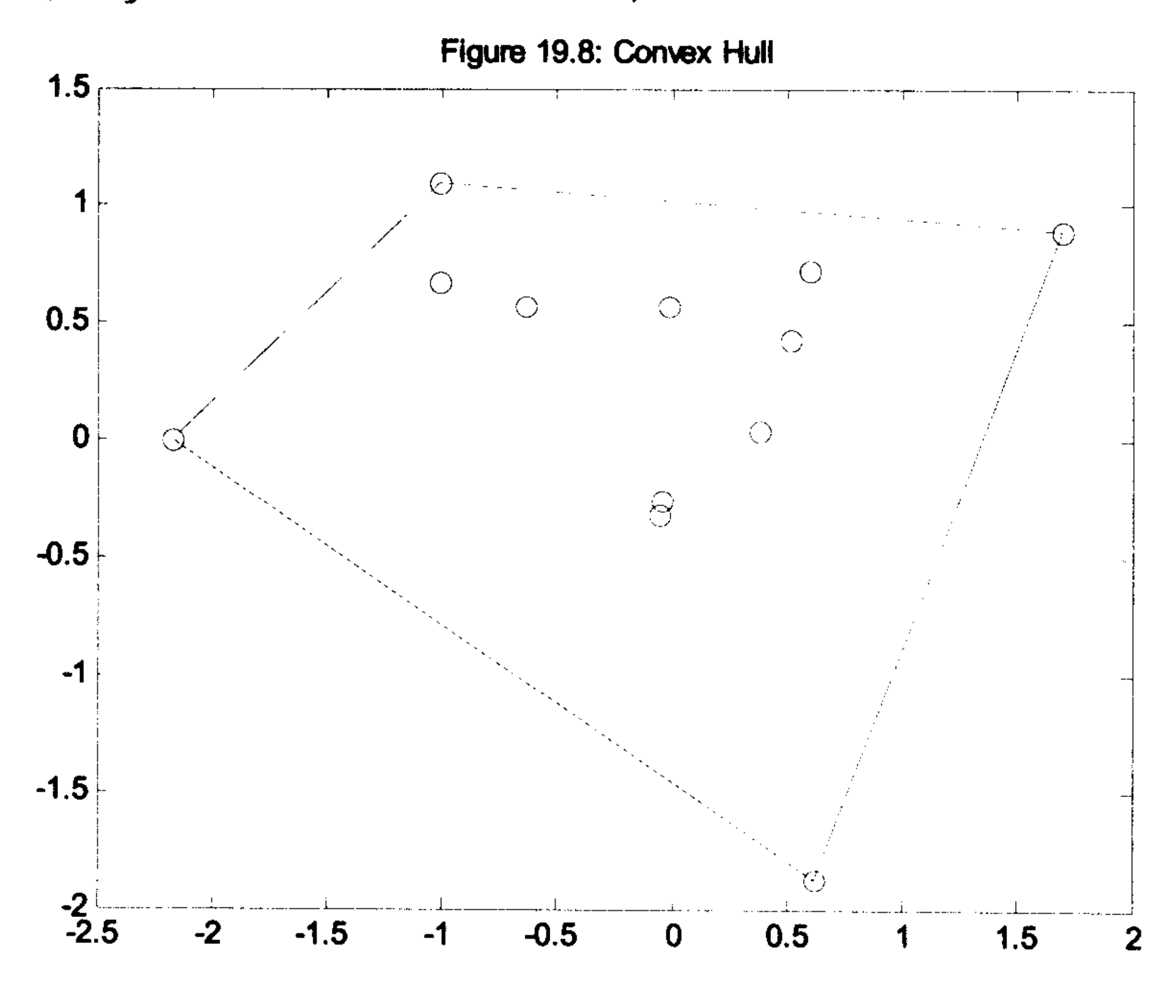


图 19.8　分散数据的凸多边形边界

分散数据的另一个分析方法是用所谓的 Voronoi 多边形分隔出最接近于一个分散点的区域。在 Matlab 中，这些分隔区域是用函数 voronoi 画出来的，例如，下面的代码对上述分散数据进行了 voronoi 多边形分隔：

```
>> voronoi(x,y)
>> title('Figure 19.9: Voronoi Diagram')
```

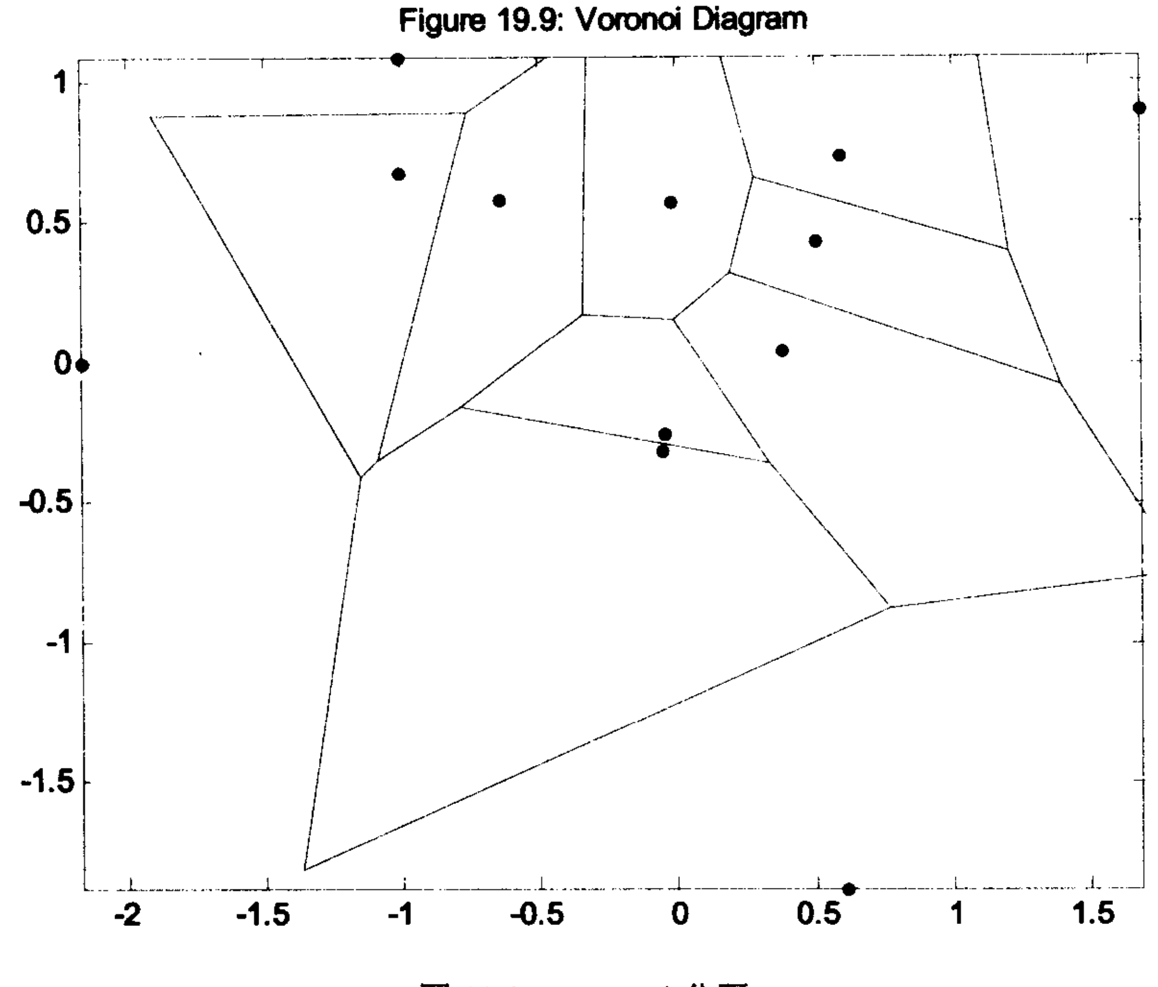


图 19.9 Voronoi 分隔

最后，用户可以使用 griddata 函数在一个 Delaunay 三角形中对分散数据进行插值以便得到用户绘图时所需要的插值点。特别地，在用户使用诸如 surf 以及其他的标准绘图函数对分散数据进行绘图时，必须首先使用该方法进行数据插值。因为这些绘图程序都要求用户提供沿两个坐标轴分布的完整的数据点信息，而不只需要分散数据。例如，我们在绘制地图时，首先用 Delaunay 三角形标志出测量得到的特定的分散点，然后利用函数 griddata 根据这些信息构建对这幅地图其他部分的估计，并利用数据插值在两个坐标方向上将估计数据填充进用户指定的方形区域内。下面给出了一个简单例子来解释上述内容：

```
>> z = rand(1,12); % now use some random z axis data
>> xi = linspace(min(x),max(x),30);     % x interpolation points
>> yi = linspace(min(y),max(y),30);     % y interpolation points

>> [Xi,Yi] = meshgrid(xi,yi);           % create grid of x and y

>> Zi = griddata(x,y,z,Xi,Yi);          % grid the data at Xi,Yi points

>> mesh(Xi,Yi,Zi)
>> hold on
>> plot3(x,y,z,'ko')    % show original data as well
>> hold off
>> title('Figure 19.10: Griddata Example')
```

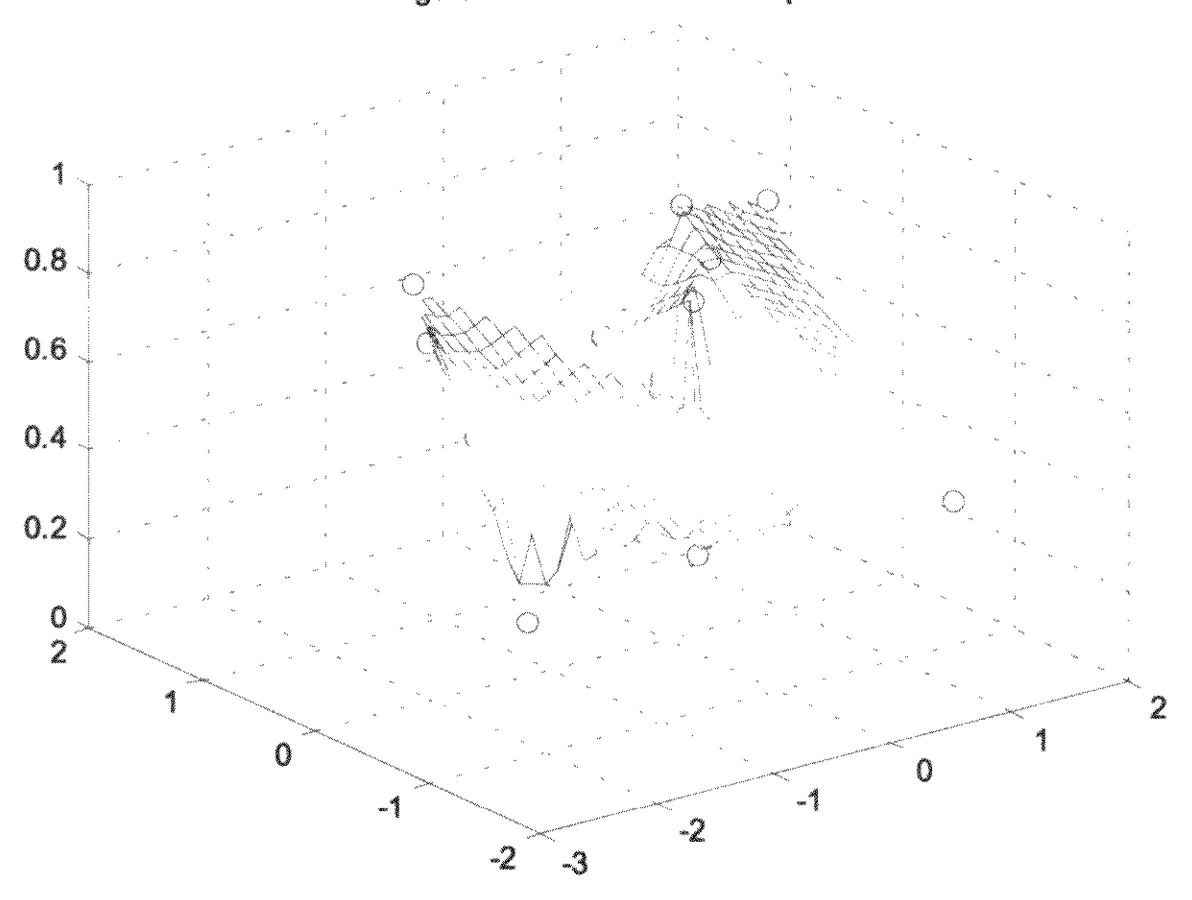


图 19.10　Griddata 函数的应用示例

在上例中，首先生成了 12 个分散数据点，然后在 x-y 平面生成 30×30 的格栅，然后利用 x、y、z 的数据进行三角形线性插值来填充栅格上的数据点，其中 30×30 的数组 Zi 就包含了这些栅格上的数据，最后进行绘图。

和其他插值函数一样，griddata 函数也支持下面代码中给出的几种插值方法：

```
>> Zi = griddata(x,y,z,Xi,Yi,'linear') % same as above(default)
>> Zi = griddata(x,y,z,Xi,Yi,'cubic') % triangle based cubic interpolation
>> Zi = griddata(x,y,z,Xi,Yi,'nearest') % triangle based nearest neighbor
```

另外，griddata 函数也具有相应的多维插值函数，这些函数这里不作阐述，它们将在下一节的汇总表格中给出。

19.4　小结

作为本章小结，本节给出了数据插值函数的汇总表：

插值函数	描述
convhull	返回分散数据的凸面边界
convhulln	返回 n 维分散数据的凸面边界
delaunay	Delaunay 三角分析
delaunay3	三维 Delaunay 三角分析
delaunayn	n 维 Delaunay 三角分析
dsearch	查找 Delaunay 三角形中与某一分散点最邻近的点
griddata	二维方形栅格数据插值

（续表）

插值函数	描述
griddata3	三维方形栅格数据插值
griddatan	n 维方形栅格数据插值
interp1	一维数据插值
interp1q	一维快速数据插值（不进行错误校验）
interp2	二维数据插值
interp3	三维数据插值
interpft	使用 FFT 方法进行一维插值
interpn	n 维数据插值
meshgrid	产生三维函数的 X 和 Y 轴的索引矩阵
ndgrid	产生多维函数的索引数组
tetramesh	绘制四面体网格
trimesh	绘制三维三角形网格
triplot	绘制二维三角形网格
trisurf	绘制三维三角形表面
tsearch	在二维分散数据中寻找包含一个数据点的 Delaunay 三角形
tsearchn	在 n 维分散数据中寻找包含一个数据点的 Delaunay 三角形
voronoi	对二维分散数据进行 Voronoi 多边形分隔
voronoin	对 n 维分散数据进行 Voronoi 多边形分隔

Chapter 20

多　项　式

在 Matlab 中，处理多项式是一件非常简单的事情，借助 Matlab 提供的函数，用户很容易对多项式进行积分、微分以及求根的操作。但有一点需要注意，当读者利用 Matlab 处理高阶（大于 10 阶）多项式时，会遇到不少计算上的麻烦，因此要格外小心。

20.1　多项式的根

求多项式的根（即使多项式等于 0 的解）是进行科学研究经常遇到的问题。在 Matlab 中，一个多项式是用多项式的系数行向量表示的，向量中的系数按照其所对应的自变量的阶次的降序进行排列。例如，多项式 $x^4-12x^3+25x+116$ 在 Matlab 中将用下面的行向量表示：

```
>> p = [1   -12    0    25    116]
p =
     1    -12     0     25    116
```

注意，上面的多项式中自变量 x 的二次方（即 x^2）不存在，则需要在系数向量中的相应元素位置上输入 0，来告诉 Matlab 该阶次的自变量不存在，用户已可以通过专门声明告诉 Matlab 这一信息。当利用系数行向量表示多项式后，就很容易利用函数 roots 求多项式的根。例如，我们可以使用下面的代码求上述多项式的根：

```
>> r =roots(p)
r =
    11.7473
    2.7028
    -1.2251 + 1.4672i
    -1.2251 - 1.4672i
```

在 Matlab 中，多项式和多项式的根都是用向量表示的，因此为了对它们加以区别，Matlab 通常将多项式表示为行向量，将多项式的根表示为列向量。

如果知道一个多项式的根，能不能构建相应的多项式呢？答案是肯定的。Matlab 提供了 poly 函数从一个根向量构建一个多项式向量。例如，我们可以利用下面的代码由 r 向量恢复原来的多项式：

```
>> pp = poly(r)
```

```
pp =
        1        -12      -1.7764e-014        25       116
>> pp(abs(pp)<1e-12) = 0        % change small element to zero!
pp =
        1        -12      0                   25       116
```

由于 Matlab 在进行数据处理时存在截断误差，因此，poly 函数的返回值有可能在该出现 0 的位置出现了一个非常接近 0 的数（如上例中的第三个系数），有时还会使某些系数带有一个很小的虚部。因此，为了保险起见，建议用户对 poly 函数的输出结果再进行一次处理，从而消除有可能出现的数据错误。例如，用户可以通过比较将绝对值极小的数强制置 0，或利用 real 函数将实部从结果中提取出来，消除错误虚部的影响。

20.2 多项式乘法

由于多项式的乘法实际上就是多项式系数向量之间的卷积运算，我们可以很简单地使用 Matlab 提供的卷积函数 conv 来完成多项式的乘法。例如，我们要求两个多项式 $a(x)=x^3+2x^2+3x+4$ 和 $b(x)=x^3+4x^2+9x+16$ 的乘积，可以采用下面的代码：

```
>> a = [1 2 3 4];   b = [1 4 9 16];
>> c = conv(a,b)
c =
    1    6    20   50   75   84   64
```

从上面的结果可知，两个多项式的乘积结果为：$c(x)=x^6+6x^5+20x^4+50x^3+75x^2+84x+64$，这与数学运算结果是一致的。

如果用户要执行多个多项式之间的乘法运算，需要重复使用 conv 函数。

20.3 多项式加法

Matlab 没有提供专门的函数执行多项式的加法。如果两个多项式向量长度相等，则多项式加法就是将两个多项式向量直接相加。例如，我们可以利用下面的代码将上节的两个多项式相加：

```
>> d = a + b
d =
    2    6    12   20
```

相加结果为 $d(x)=2x^3+6x^2+12x+20$。

当两个多项式的阶次不同时，其系数向量的长度也不同，这时需要先将低阶多项式的系数向量前边补上足够的 0 以便使它和高阶多项式具有相同的长度，然后再执行加法运算。例如，我们要将 20.2 节的多项式 c 和上面的 d 相加，可以使用下面的代码：

```
>> e = c + [0 0 0 d]
e =
    1    6    20   52   81   96   84
```

可见，两个多项式相加的结果为 $e(x)=x^6+6x^5+20x^4+52x^3+81x^2+96x+84$。

执行加法前之所以在向量的前面补 0 而不在后面补 0，是因为我们需要补足的是低阶多项式缺少的高阶次成分，而高阶系数位于系数向量的前面。

每次进行多项式加法都进行补零显然是一件非常麻烦的事，因此，建立一个 M 函数文件用于自动完成多项式加法是十分必要的。下面就是一个作者自己编写的一个多项式加法的 M 函数文件：

```
function p=mmpadd(a,b)
%MMPADD Polynomial Addition.
% MMPADD(A,B) adds the polynomials A and B.

if nargin<2
   error('Not Enough Input Arguments.')
end

a=reshape(a,1,[]);          % make sure inputs are polynomial row vectors
b=b(:).';                   % this makes a row as well

na=length(a);               % find lengths of a and b
nb=length(b);

p=[zeros(1,nb-na) a]+[zeros(1,na-nb) b];  % pad with zeros as necessary
```

现在，我们可以使用 mmpadd 函数来完成上面的加法运算，如下面的代码所示：

```
>> f = mmpadd(c,d)
f =
    1    6    20   52   81   96   84
```

我们发现，两种方法得到了相同的结果。

另外，利用 mmpadd 函数也可以执行多项式的减法。例如，下面的代码执行多项式 c 减去多项式 d 的操作：

```
>> g = mmpadd(c,-d)
g =
    1    6    20   48   69   72   44
```

由上可知，两个多项式相减的结果是 $g(x)=x^6+6x^5+20x^4+48x^3+69x^2+72x+44$。

20.4 多项式除法

多项式除法虽然没有乘法和加法常用，但在某些特殊情况下，我们也需要用一个多项式去除以另一个多项式。在 Matlab 中，多项式除法是由函数 deconv 实现的，例如，下面的代码执行多项式 c 除以多项式 b 的操作：

```
>> [q,r] = deconv(c,b)
q =
    1    2    3    4
r =
    0    0    0    0    0    0    0
```

上面的函数调用中，q 用来存储相除后的商，r 用来存储相除后的余数。由于 c 就是 a 和 b 的乘积，因此，余数 r 等于全 0 向量，而 q 等于 a。下面给出了除不尽的例子：

```
>> [q,r] = deconv(f,b)
q =
     1     2     3     6
r =
     0     0     0     0    -2    -6   -12
```

从结果可以看出，当用 f 除以 b 时，将得出商为 $q(x)= x^3+2x^2+3x+6$，余数为 $r(x)=-2x^2-6x-12$。由于余数（本例中为 r）通常都与除数和被除数中较长的那个向量（本例是 f）等长，因此，r 的前面补充了 4 个 0。

20.5 多项式的微分和积分

在 Matlab 中，多项式的微积分执行起来非常简单，只要简单调用两个函数 polyder（完成微分操作）和 polyint（完成积分操作）即可。

下面的代码完成 20.3 节的多项式 g 的微分（求导）：

```
>> g % recall polynomial
g =
     1     6    20    48    69    72    44
>> h = polyder(g)
h =
     6    30    80   144  138   72
```

在执行一个多项式的积分时，用户需要为函数 polyint 提供一个积分常数。例如，下面的代码完成了 h 的积分，从而又得到了 g：

```
>> polyint(h,44)     % get g back from h=polyder(g)
ans =
     1     6    20    48    69    72    44
```

20.6 多项式求值

在 Matlab 中，除了可以对多项式进行加法、减法、乘法、除法以及微分运算外，用户也可以利用函数 polyval 对多项式进行求值。例如，下面的代码对多项式 $x^3+4x^2-7x-10$ 在[-1, 3]之间求值并画出了相应的曲线：

```
>> p = [1 4 -7 -10];           % the polynomial
>> x = linspace(-1,3);         % evaluation points
>> v = polyval(p,x);           % evaluate p at points in x
>> plot(x,v)                   % plot results
>> title('Figure 20.1: x{^3} + 4x{^2} - 7x -10')
>> xlabel('x')
```

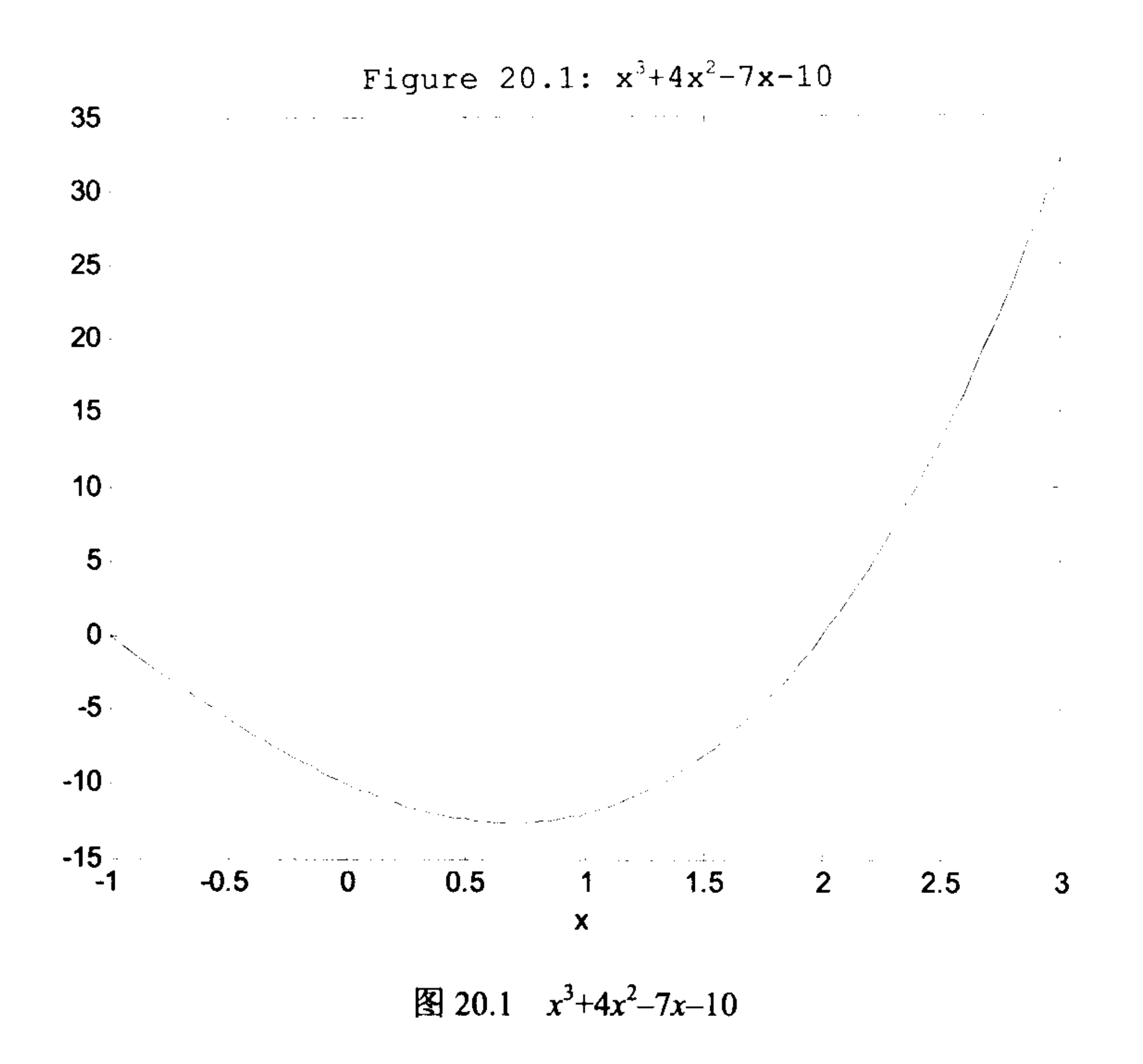


图 20.1 $x^3+4x^2-7x-10$

20.7 有理多项式

在很多时候，用户需要处理两个多项式的比值，例如，系统传递函数和函数的 Pade 估计，这样的多项式通常称为有理多项式，也叫比例多项式。在 Matlab 中，对有理多项式的处理是通过分别处理其分子和分母来完成的。有理多项式的一个重要特征是零点和极点分布，其中零点为其分子多项式的根，极点为其分母多项式的根。下面的代码创建了一个有理多项式 $n(x)/d(x)$，并求出了它的零极点：

```
>> n = [1 -10 100]          % a numerator
n =
    1   -10     100
>> d = [1 10 100 0]         % a denominator
d =
    1   10      100     0
>> z = roots(n)             % the zeros of n(x)/d(x)
z =
        5 +         8.6603i
        5 -         8.6603i
>> p = roots(d)             % the poles of n(x)/d(x)
p =
         0
        -5 +        8.6603i
        -5 -        8.6603i
```

Matlab 还提供了其他一些函数来处理有理多项式。例如，上面的有理多项式的导数可以使用 polyder 函数来计算，如下面的代码所示：

```
>> [nd,dd] = polyder(n,d)
nd =
        -1   20   -100     -2000    -10000
dd =
    Columns 1 through 6
         1   20    300      2000     10000       0
    Column 7
         0
```

在调用 polyder 函数时，需要同时为其提供分子和分母多项式，并最终生成微分后的分子和分母多项式。

另外，我们可以利用 residue 函数求有理多项式 $n(x)/d(x)$的部分分式展开，代码如下：

```
>> [r,p,k] = residue(n,d)
r =
    9.7954e-17 +    1.1547i
    9.7954e-17 -    1.1547i
         1
p =
        -5 +        8.6603i
        -5 -        8.6603i
         0
k =
    []
```

如上面的例子所示，residue 函数一般返回 3 个变量：部分分式展开系数 r、极点 p 和余数多项式 k。由于本例中分子的阶次小于分母，因此不存在余数多项式，则 k 为[ ]。根据上面的结果，$n(x)/d(x)$的部分分式展开为：

$$\frac{n(x)}{d(x)}=\frac{1.1547i}{x+5-8.6603i}+\frac{-1.1547i}{x+5+8.6603i}+\frac{1}{x}$$

将上面的计算结果作为输入参数，再次使用 residue 函数就可以求得原来的比例多项式。

```
>> [nn,dd] = residue(r,p,k)
nn =
        1        -10       100
dd =
        1         10       100      0
```

从上面的例子可以看到，residue 函数将根据使用了多少个输入和输出参数，来判断执行什么样的操作。也就是说，当输入参数为 3 个、输出参数为 2 个时，将执行合并多项式的操作；当输入参数为 2 个、输出参数为 3 个时，将执行部分分式展开的操作。

20.8 曲线拟合

曲线拟合是进行数据分析时经常遇到的问题，它指根据一组或多组测量数据找出一条数学上可描述的曲线的过程。这条曲线有时候将穿过测量的数据点，而有时候将会非常接近于但不会穿过测量的数据点。评价一条曲线是否准确地描述了测量数据的最通用的方法，

是看测量数据点与该曲线上对应点之间的平方误差是否达到最小，这种曲线拟合的方法称为最小二乘曲线拟合。原则上，我们可以选择任何一组基本函数实现最小二乘曲线拟合，其中使用多项式是最简单最常用的方法。

Matlab 提供了函数 polyfit 用于实现最小二乘多项式曲线拟合。下面我们通过一个简单的例子看一下该函数的用法。假设我们要对下面的数据进行曲线拟合：

```
>> x = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1];
>> y = [-.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.30 11.2];
```

polyfit 函数除了需要上面的测量数据外，还需要用户希望拟合的多项式曲线的最高阶数。当我们将该阶数设为 1 时，表示要进行最佳直线拟合（又叫线性回归）；当我们将该阶数设为 2 时，表示要为上述数据找出一个二次多项式拟合曲线。例如，我们要选择一个二次多项式进行拟合，可以使用下面的代码：

```
>> n = 2;
>> p = polyfit(x,y,n)
p =
   -9.8108     20.1293     -0.0317
```

由于我们要求的是一个二阶多项式曲线拟合，因此 polyfit 的输出就是一个含有三个元素的行向量，分别表示该拟合曲线的二阶、一阶系数和常数项。由 p 可知，我们求出的结果为：$y(x)=-9.8108x^2+20.1293x-0.0317$。为了观察曲线拟合的效果，我们用下面的代码将该曲线和原来的数据画在图 20.2 中：

```
>> xi = linspace(0,1,100);
>> yi = polyval(p,xi);
>> plot(x,y,'-o',xi,yi,'--')
>> xlabel('x'),ylabel('y=f(x)')
>> title('Figure 20.2: Second Order Curve Fitting')
```

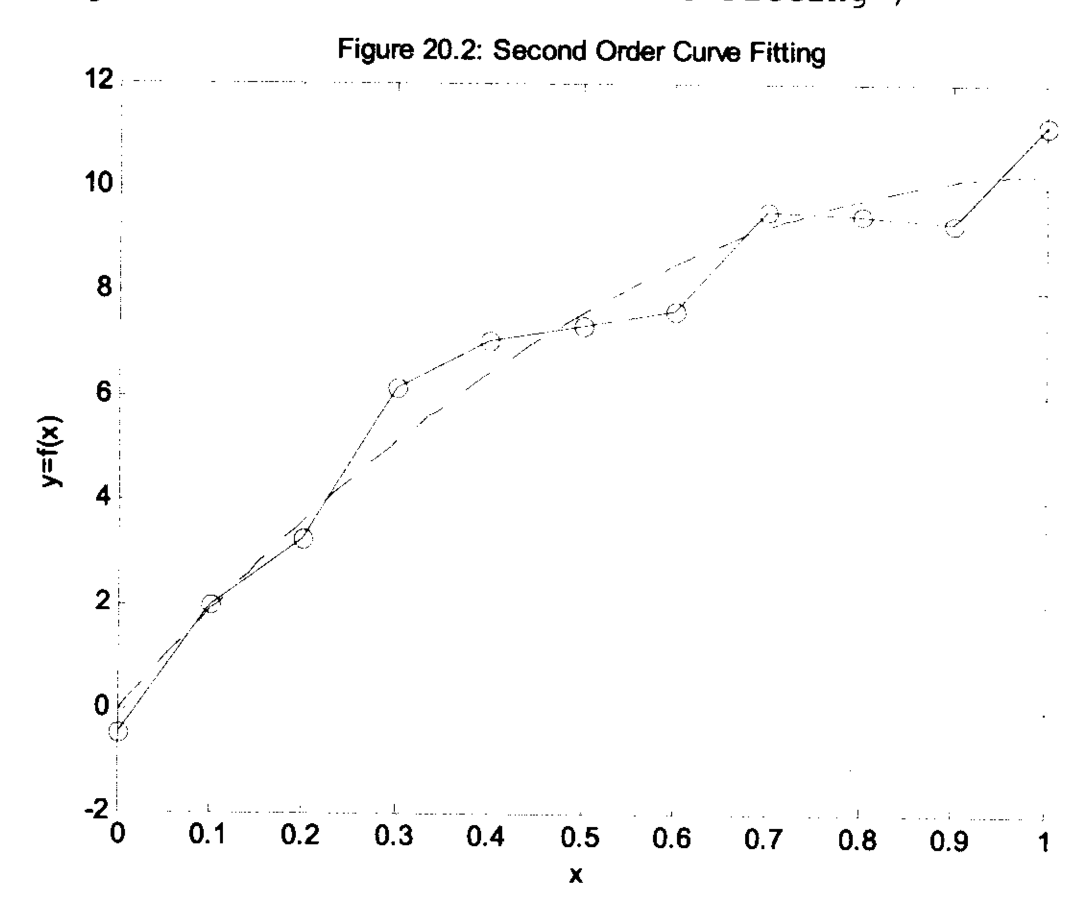


图 20.2　二阶曲线拟合示例

在图 20.2 中，原数据点用圆圈（'o'）表示，二阶拟合曲线用虚线表示。为了方便比较，我们将数据点用直线连接起来。

实际上，在进行曲线拟合时对多项式阶次的选择是任意的。从数学原理上讲，两个点可以惟一定义一条直线（或一阶多项式）；3 个点可以惟一定义一个二次曲线（或二阶多项式）；依此类推，n+1 个数据点可以惟一定义一个 n 次曲线（或 n 阶多项式）。由于上例中共有 11 个数据点，因此这些点可以惟一定义一个 10 阶多项式。虽然高阶的多项式可以更准确地拟合数据（这里我们仅指该曲线与我们给定的这些数据之间的均方误差最小），但我们在进行曲线拟合时，并不需要采用太高阶的多项式，这主要基于以下原因：①越是高阶的多项式其数值特性越差，计算起来也越耗时。②随着多项式阶次的升高，拟合的曲线变得越来越不平滑，通常会出现用户不愿意看到的局部波形（这一点在后面的图中表现比较明显）。③由于用户数据本身的近似性，因此用户在进行数据拟合时没有必要仅仅考虑使拟合的曲线无限接近数据点，而要在曲线的阶次和均方误差之间综合考虑，因为越是高阶的多项式在物理实现时越困难。不过，为了进行比较，我们来选择一个 10 阶的多项式对上述数据进行拟合，代码如下：

```
>> pp = polyfit(x,y,10);
>> pp.'      % display polynomial coefficients as a column
ans =
    -4.6436e+005
     2.2965e+006
    -4.8773e+006
     5.8233e+006
    -4.2948e+006
     2.0211e+006
    -6.0322e+005
      10896e+005
         -10626
         435.99
         -0.447
```

由于我们采用的是 10 阶拟合，因此 polyfit 的返回参数中含有 11 个元素。另外，这些系数之间存在很大的差异（最小的数-0.447 和最大的数 5.8233e+006 之间竟然有 7 个数量级的差异！），这给用户的后续处理带来了很大麻烦。另外，这些系数以正负交替的形式出现，以保证曲线在一定范围内的平稳变化（若都是正的，则曲线将会出现陡升）。为了比较多项式阶次对曲线拟合的影响，我们使用下面的代码将两次拟合的结果绘制在图 20.3 上：

```
>> y10 = polyval(pp,xi);    % evaluate 10th order polynomial
>> plot(x,y,'o',xi,yi, '--',xi,y10) % plot data
>> xlabel('x'),ylabel('y=f(x)')
>> title('Figure 20.3: 2nd and 10th Order Curve Fitting')
```

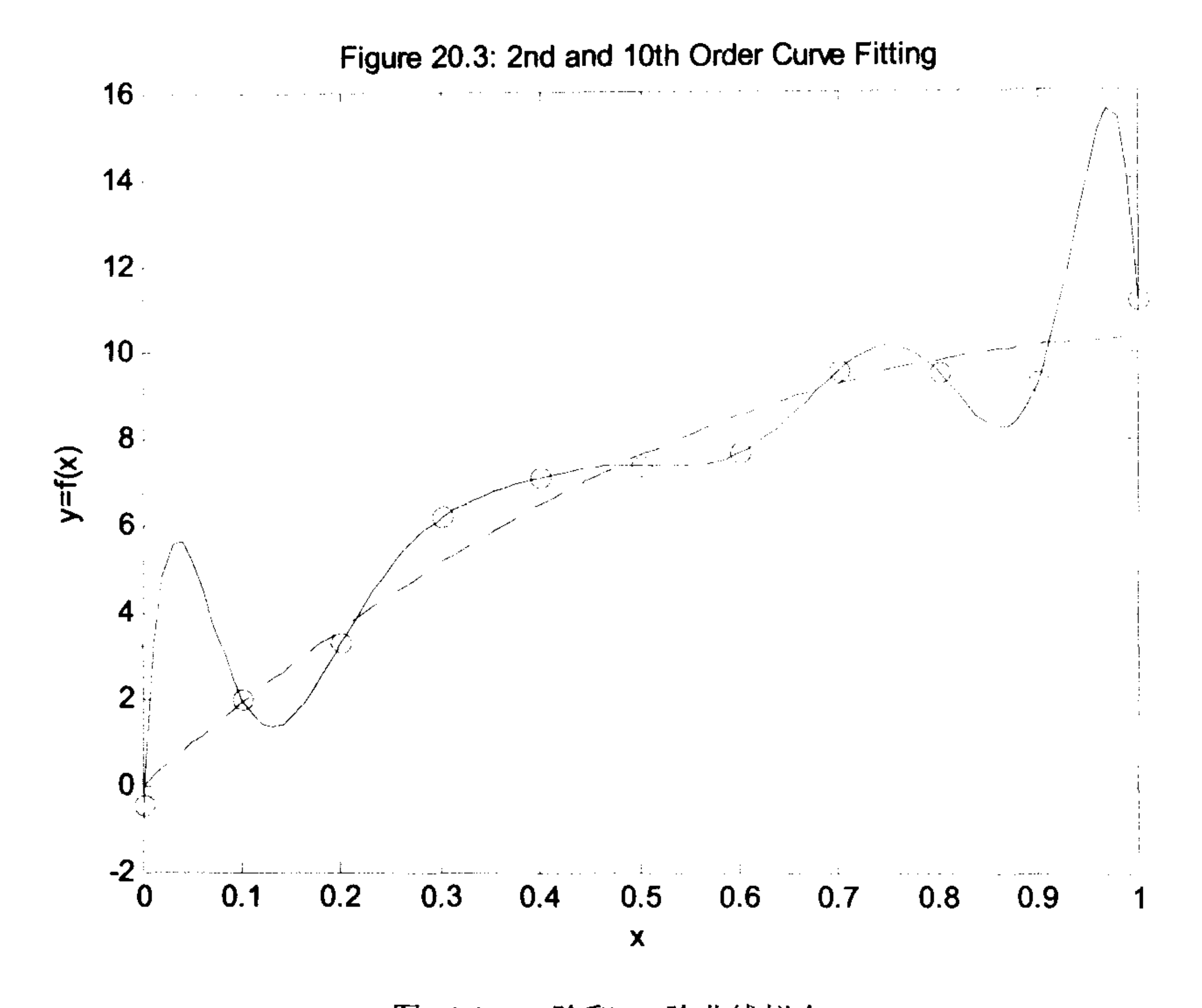


图 20.3 二阶和 10 阶曲线拟合

在图 20.3 中，原始数据用圆圈'o'标出，二阶曲线拟合用虚线表示，10 阶曲线拟合用实线表示。从该图我们可以看到，10 阶拟合曲线在图形的左右两端出现了用户不愿意看到的波浪型曲线，另外 10 阶拟合曲线在平滑性上远远不及二阶曲线。

Chapter 21

三次样条函数

众所周知，采用高阶多项式进行插值通常会带来不正常的结果。因此，人们开始寻求其他的解决办法，这其中，三次样条函数得到了普遍应用。Matlab 提供了函数 spline、ppval、mkpp 和 unmkpp 来实现三次样条函数插值。在这些函数中，Matlab 着重介绍了 spline 函数，对其他函数只提供了相应的帮助文档。本章主要介绍了三次样条函数的基本特性；讨论了三次样条函数中的一种称为“分段立方厄密插值多项式”的方法，同时还介绍了计算厄密分段多项式的一个基本函数：pchip。

21.1 基本特性

立方样条函数均使用三次多项式来近似估计每对数据点之间的曲线，这样的数据点数学上称为断点。因为两个点只能惟一确定一条直线，因此这两个点之间的曲线可以用无穷多个三次多项式来逼近。因此，为了便于实现，必须对这些三次多项式添加额外的约束条件使得曲线估计具有惟一解。首先，必须保证与断点相连的三次多项式经过断点，这就需要限制这些三次多项式的一阶和二阶导数值，以便使其能取到断点处原始数据的值，当断点处的多项式确定后，内部的三次多项式也都会随之确定。其次，三次多项式在断点处的斜率和曲率都必须是连续的。由于与第一个和最后一个断点相连接的三次多项式没有与之相邻的多项式，这样就必须依靠其他方法来对这两个多项式进行约束。一种最常用的方法（也是函数 spline 采用的默认方法）就是采用非结（not-a-knot）条件，该条件规定与第一个断点相连的多项式和后面相邻的多项式的三阶导数相同，与最后一个断点相连的多项式和前面相邻的多项式的三阶导数相同。

根据上边的描述，读者可能会认为，寻找三次样条多项式需要求解一个庞大的线性方程组。实际上不是这样的。假如给定 n 个断点，需要找出 n-1 个三次多项式来估计每对断点之间的曲线，其中每个三次多项式有 4 个未知系数，这样，需要求解的方程组就包含 4×(n-1)个未知数。不要忘了，我们还有许多约束条件没有用上。当我们将所有的约束条件应用到每一个三次多项式以后，该问题就变成了通过 n 个方程求解 n 个未知数的一个方程组（这一结论读者可以根据上面的约束条件推导，或参考相关的数学文献）。例如，如果有 50 个断点，估计这 50 个断点之间的曲线就相当于求解含有 50 个未知数的 50 个方程。

虽然这看起来计算量仍很大，但 Matlab 可以方便地采用稀疏矩阵求解，使得整个求解过程变得简单易行。

21.2　分段多项式

spline 函数可用于计算分段多项式。该函数利用数据 x 和 y 以及期望向量 xi，找出拟合 x 和 y 的三次样条插值多项式，然后通过计算多项式的值求出与每个 xi 值相对应的 yi。该函数功能与下面的语句一样：yi=interp1(x,y,xi,'spline')。下例给出了 spline 函数的一个应用：

```
>> x = 0:12;
>> y = tan(pi*x/25);
>> xi = linspace(0,12);
>> yi = spline(x,y,xi);
>> plot(x,y,'o',xi,yi)
>> title('Figure 21.1: Spline Fit')
```

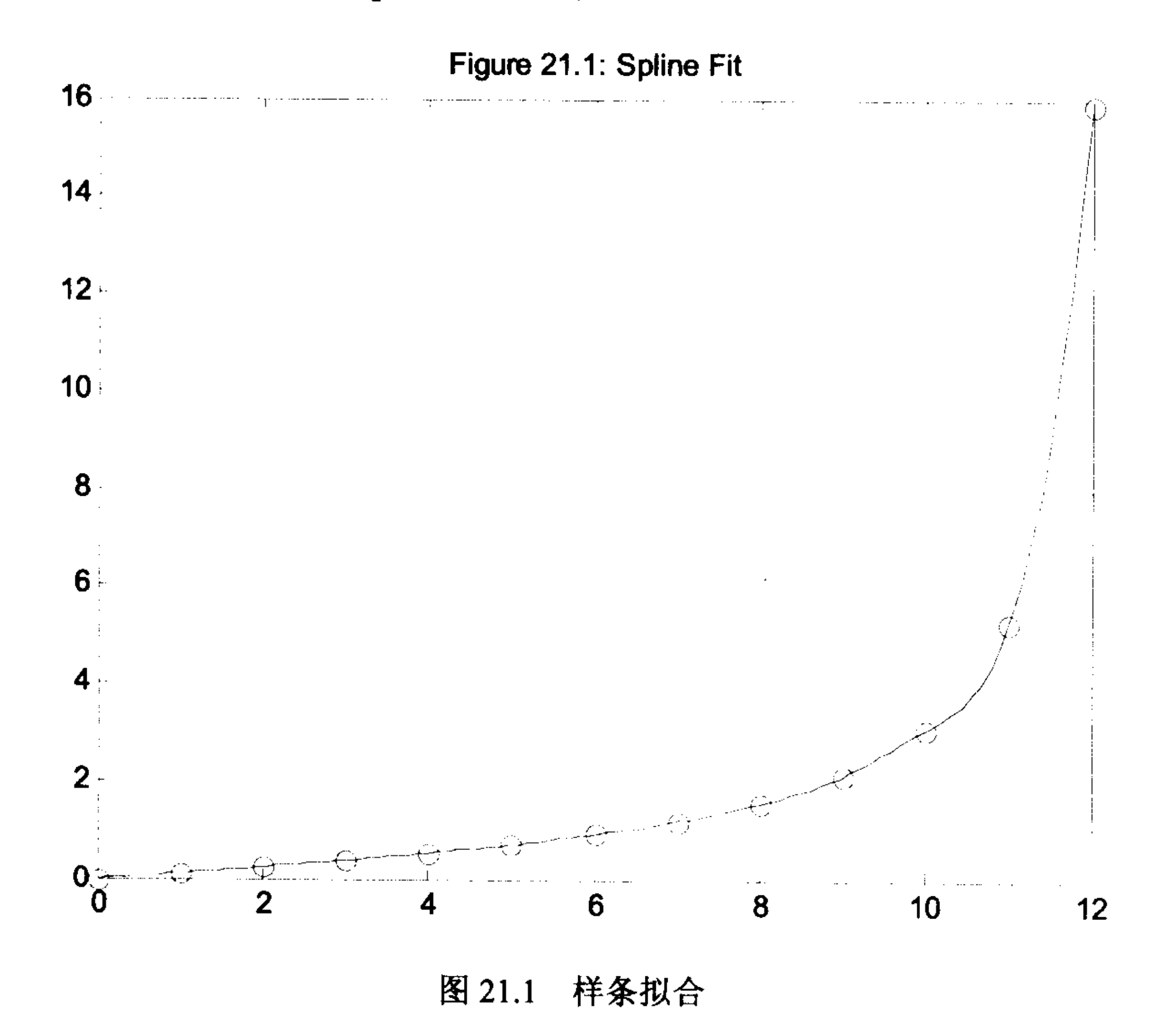


图 21.1　样条拟合

上例利用 spline 函数和一组离散数据拟合出了 tan 函数的曲线。

当用户只需要一组插值数据时，可以采用上面的 spline 调用方法，但如果用户需要多组插值数据时（比如同一区间中的不同精度的插值数据），就没有必要反复计算相同的三次样条系数。在这种情况下，用户可以采用下面的方式调用 spline 函数：

```
>> pp = spline(x,y)
pp =
```

```
     form: 'pp'
   breaks: [0 1 2 3 4 5 6 7 8 9 10 11 12]
    coefs: [12×4 double]
   pieces: 12
    order: 4
      dim: 1
```

此时，spline 函数将返回一个包含三次样条函数的分段多项式格式（简称 pp 格式）的结构体 pp。pp 包含了用户执行三次样条插值所需的所有信息。另外，pp 也可以在 Matlab 中提供的样条工具箱中使用。函数 ppval 可以利用 pp 反复计算三次样条插值，不再需要每次都计算三次样条系数，因为三次样条系数已作为输入参数（包存在 pp 中）传递给了 ppval 函数。例如，下面的代码执行了与前面相同的插值操作：

```
>> yi = ppval(pp,xi);
```

利用上面的方法，用户可以很方便的在一个更精确的区域内重新计算三次样条插值操作。例如，下面的代码在[10,12]区间计算更加精确的三次样条值：

```
>> xi2 = linspace(10,12);
>> yi2 = ppval(pp,xi2);
```

上面的方法还可以进行三次多项式计算区域之外的插值。当数据出现在最后一个断点之后或第一个断点之前时，ppval 将分别利用最后一个和第一个三次多项式计算插入点的值。例如，下面的代码利用 pp 计算[10,15]之间的数据插值（注意 pp 的有效计算范围是[0,12]）：

```
>> xi3 = 10:15;

>> yi3 = ppval(pp,xi3)
yi3 =
    3.0777    5.2422    15.8945    44.0038    98.5389    188.4689

>> yi4 = ppval(xi3,pp) % can be called with arguments reversed
yi4 =
    3.0777    5.2422    15.8945    44.0038    98.5389    188.4689
```

最后一条语句表明，ppval 对输入参数的顺序没有要求。因此，用户就可以创建一个函数句柄，然后将其作为输入参数传递给一个函数用于完成用户需要的某一特定操作。例如，下面的代码计算由 pp 计算出的三次样条曲线在[0,10]之间的投影区域的面积：

```
>> quad(@ppval,0,10,[],[],pp)
ans =
        9.3775
```

有关 quad 函数的用法请读者参考本书第 24 章。

由上面的例子可以看出，pp 格式结构体（如 pp）是 Matlab 提供给用户的一个十分有用的数据结构，因为该结构体保存了进行三次样条插值所需的诸如断点、多项式系数以及分段数等重要的信息。有时候，用户在计算一个三次样条表达式时，需要将所需的数据从上述结构体中提取出来，这时，需要用到 unmkpp 函数。例如，下面的代码利用 unmkpp 函数将 pp 中的数据信息提取到不同的变量中：

```
>> [breaks,coefs,npolys,ncoefs,dim] = unmkpp(pp)
breaks =
   Columns 1 through 12
      0     1     2     3     4     5     6     7     8     9    10    11
   Column 13
     12
coefs =
     0.0007   -0.0001    0.1257         0
     0.0007    0.0020    0.1276    0.1263
     0.0010    0.0042    0.1339    0.2568
     0.0012    0.0072    0.1454    0.3959
     0.0024    0.0109    0.1635    0.5498
     0.0019    0.0181    0.1925    0.7265
     0.0116    0.0237    0.2344    0.9391
    -0.0083    0.0586    0.3167    1.2088
     0.1068    0.0336    0.4089    1.5757
    -0.1982    0.3542    0.7967    2.1251
     1.4948   -0.2406    0.9102    3.0777
     1.4948    4.2439    4.9136    5.2422
npolys =
     12
ncoefs =
     4
dim =
     1
```

上例中，breaks 为断点，coefs 是三次多项式系数矩阵，其第 i 行是第 i 个三次多项式的系数，npolys 是多项式的个数，ncoefs 是每个多项式系数的个数，dim 是样条的维数。请注意，pp 格式结构体是一个通用结构，它可以表示任何阶数的多项式，不只局限在三阶。这一通用性使得它在后面将要讲到的样条积分和微分中仍具有重要的作用。

在 Matlab 6 之前的版本中，pp 格式是保存在一个数值型数组而不是结构体中的。用户仍可以使用函数 unmkpp 将 pp 格式中的数据从数值型数组中提取出来。虽然数组形式的 pp 格式没有结构体形式的 pp 格式简单实用，但由于有 unmkpp 函数的支持，用户一样可以轻松地处理以前的版本创建的 pp 格式数组。

给定一组断点和系数矩阵，利用函数 mkpp 可以创建一个 pp 格式结构体，如下例所示：

```
>> pp = mkpp(breaks,coefs)
pp =
     form:'pp'
   breaks:[0 1 2 3 4 5 6 7 8 9 10 11 12]
    coefs:[12×4 double]
   pieces:12
    order:4
      dim:1
```

上例中，由于矩阵 coefs 的维数已经指定了 npolys 和 ncoefs，因此用户不需要给 mkpp 提供这两个参数。

21.3 三次厄密多项式

当被插值的原始数据代表一个平滑函数时，三次样条函数能够得到非常恰当的解。但是，当原始数据代表一个非平滑函数时，三次样条函数就有可能预测出并不存在的极值（最小值和最大值），从而破坏原函数的单调性。因此，对于非平滑数据，需要采用其他的分段多项式插值函数。在 Matlab 中，函数 pchip 是完成这一任务的首选。关于该函数的属性和用法请读者参看下面给出的该函数的帮助文档：

```
>> help pchip

 PCHIP  Piecewise Cubic Hermite Interpolating Polynomial.
   PP = PCHIP(X,Y) provides the piecewise polynomial form of a certain
   shape-preserving piecewise cubic Hermite interpolant, to the values
   Y at the sites X, for later use with PPVAL and the spline utility UNMKPP.
   X must be a vector.
   If Y is a vector, then Y(j) is taken as the value to be matched at X(j),
   hence Y must be of the same length as X.
   If Y is a matrix or ND array, then Y(:,...,:,j) is taken as the value
   to be matched at X(j),  hence the last dimension of Y must equal length(X).
   YY = PCHIP(X,Y,XX) is the same as YY = PPVAL(PCHIP(X,Y),XX), thus
   providing, in YY, the values of the interpolant at XX.
   The PCHIP interpolating function, p(x), satisfies:
   On each subinterval,  X(k) <= x <= X(k+1),  p(x) is the cubic Hermite
      interpolant to the given values and certain slopes at the two endpoints.
   Therefore, p(x) interpolates Y, i.e., p(X(j)) = Y(:,j), and
      the first derivative, Dp(x), is continuous, but
      D^2p(x) is probably not continuous; there may be jumps at the X(j).
   The slopes at the X(j) are chosen in such a way that
      p(x) is "shape preserving" and "respects monotonicity". This means
   that,on intervals where the data is monotonic, so is p(x);
   at points where the data have a local extremum, so does p(x).
 Comparing PCHIP with SPLINE:
   The function s(x) supplied by SPLINE is constructed in exactly the same
   way,except that the slopes at the X(j) are chosen differently, namely
   to make even D^2s(x) continuous. This has the following effects.
   SPLINE is smoother, i.e., D^2s(x) is continuous.
   SPLINE is more accurate if the data are values of a smooth function.
   PCHIP has no overshoots and less oscillation if the data are not smooth.
   PCHIP is less expensive to set up.
   The two are equally expensive to evaluate.
```

下面的例子演示了 spline 函数和 pchip 函数之间的异同之处：

```
>> x = [0 2 4 5 7.5 10];    % sample data
>> y = exp(-x/6).*cos(x);

>> cs = spline(x,y);        % cubic spline
>> ch = pchip(x,y);         % cubic Hermite

>> xi = linspace(0,10);
```

```
>> ysi = ppval(cs,xi);      % interpolate spline
>> yhi = ppval(ch,xi);      % interpolate Hermite
>> plot(x,y,'o',xi,ysi,':',xi,yhi)
>> legend('data','spline','hermite')
>> title('Figure 21.2: Spline and Hermite Interpolation')
```

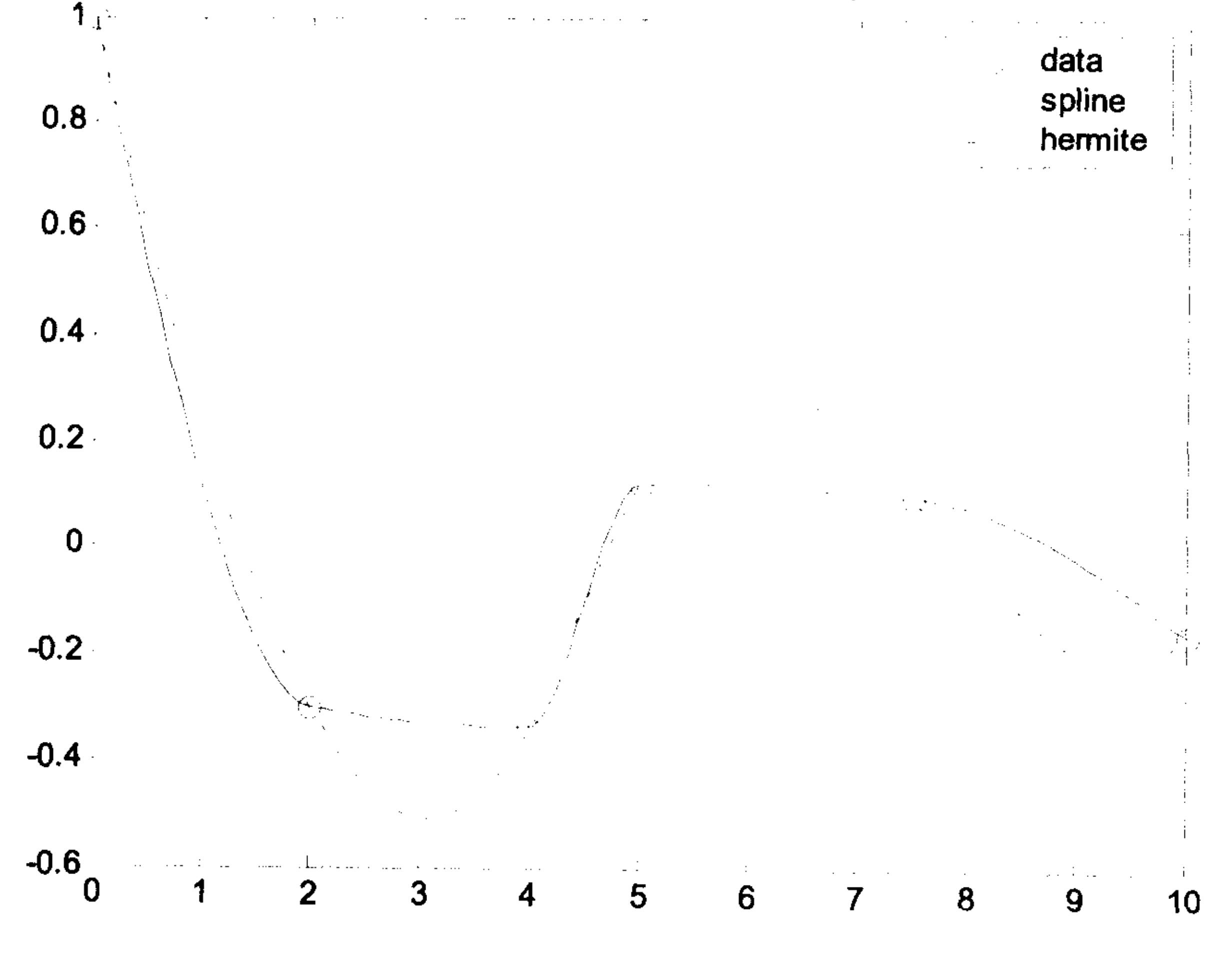


图 21.2　样条和厄密插值

由图 21.2 可以看出，厄密插值更能反映原始数据真实的变化情况。

21.4　积分

很多时候，用户都希望知道用分段多项式描述的曲线与自变量坐标轴之间的区域的面积（也称投影区域的面积）。当我们考察该面积与自变量 x 之间的关系时，通常将其表示为 x 的函数，它表示自第一个断点到 x 这一点之间的曲线到 x 轴的投影面积，其中 x 介于第一个断点和最后一个断点之间。如果分段多项式用 $y=s(x)$表示，则投影面积通常用积分的形式表示如下：

$$S(x)=\int_{x_1}^{x} s(x)\mathrm{d}x + C$$

其中，x_1 是第一个断点，C 是积分常数。由于 $s(x)$是由彼此相连的三次多项式组成的，设其第 k 个三次多项式为：

$$s_k(x)=a_k(x-x_k)^3+b_k(x-x_k)^2+c_k(x-x_k)+d_k \qquad x_k \le x \le x_{k+1}$$

则该多项式在区域 $x_k \leqslant x \leqslant x_{k+1}$ 上的投影面积为：

$$S_k\ (x)=\int_{x_k}^{x} s_k(x)\mathrm{d}x=\frac{a_k}{4}(x-x_k)^4+\frac{b_k}{3}(x-x_k)^3+\frac{c_k}{2}(x-x_k)^2+d_k(x-x_k)$$

整个分段多项式的总的投影面积将为：

$$S(x)=\sum_{i=1}^{k-1}S_i(x_{i+1})+S_k(x)$$

其中，$x_k \leqslant x \leqslant x_{k+1}$。$\sum_{i=1}^{k-1}S_i(x_{i+1})$是对每一段三次多项式的投影面积求和，它将构成$S(x)$的常数项的一部分。由于$S_k(x)$本身是一个四阶分段多项式，因此$S(x)$也是一个四阶分段多项式。

前一节讲到，pp 格式结构体可以应用于任何阶次的多项式，因此上边的分段多项式积分可以用下面的 M 文件函数 mmppint 实现：

```
function ppi=mmppint(pp,c)
%MMPPINT Cubic Spline Integral Interpolation.
% PPI=MMPPINT(PP,C) returns the piecewise polynomial vector PPI
% describing the integral of the cubic spline described by
% the piecewise polynomial in PP and having integration constant C.

if prod(size(c))~=1
   error('C Must be a Scalar.')
end
[br,co,npy,nco]=unmkpp(pp);          % take apart pp
sf=nco:-1:1;                         % scale factors for integration
ico=[co./sf(ones(npy,1),:) zeros(npy,1)]; % integral coefficients
nco=nco+1;                      % integral spline has higher order
ico(1,nco)=c;                   % integration constant
for k=2:npy                     % find constant terms in polynomials
     ico(k,nco)=polyval(ico(k-1,:),br(k)-br(k-1));
end
ppi=mkpp(br,ico);               % build pp form for integral
```

下例给出了使用上面的函数进行积分计算的一个例子：

```
>> x = (0:.1:1)*2*pi;
>> y = sin(x);              % create rough data
>> pp = spline(x,y);        % pp-form fitting rough data
>> ppi = mmppint(pp,0);     % pp-form of integral
>> xi = linspace(0,2*pi);   % finer points for interpolation
>> yi = ppval(pp,xi);       % evaluate curve
>> yyi = ppval(ppi,xi);     % evaluate integral
>> plot(x,y,'o',xi,yi,xi,yyi,'--') % plot results
>> title('Figure 21.3: Spline Integration')
```

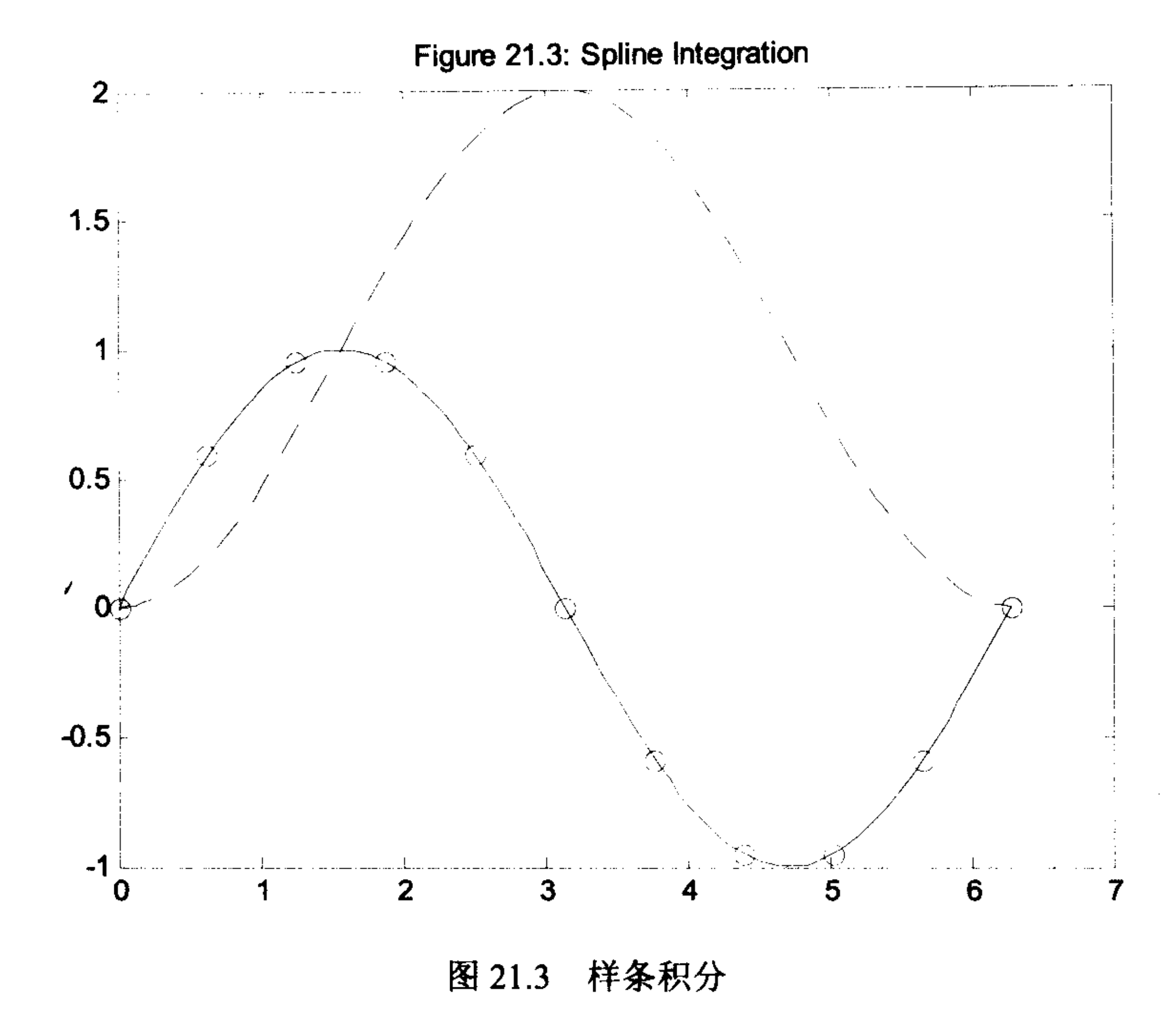


图21.3　样条积分

注意，该图证明了下面的等式成立：

$$\int_0^x \sin(x)\mathrm{d}x = 1-\cos(x)$$

21.5　微分

既然可以对分段多项式进行积分，我们同样也可以对分段多项式进行微分（即求斜率）。我们仍以前一节的多项式为例，假设第 k 个三次多项式为：

$$s_k(x)=a_k(x-x_k)^3+b_k(x-x_k)^2+c_k(x-x_k)+d_k \qquad x_k \le x \le x_{k+1}$$

那么 $s_k(x)$的微分可以写为：

$$\frac{\mathrm{d}s_k(x)}{\mathrm{d}x}=3a_k(x-x_k)^2+2b_k(x-x_k)+c_k$$

其中，$x_k \leqslant x \leqslant x_{k+1}$。与积分一样，分段多项式的微分也是一个分段多项式，只不过该分段多项式是二阶的。

我们也提供了一个 M 文件函数 mmppder 来实现分段多项式的微分计算。该函数的函数体如下所示：

```
function ppd=mmppder(pp)
%MMPPDER Cubic Spline Derivative Interpolation.
% PPD=MMPPDER(PP) returns the piecewise polynomial vector PPD
% describing the cubic spline derivative of the curve described
% by the piecewise polynomial in PP.
```

```
[br,co,npy,nco]=unmkpp(pp);             % take apart pp
sf=nco-1:-1:1;                          % scale factors for differentiation
dco=sf(ones(npy,1),:).*co(:,1:nco-1);  % derivative coefficients
ppd=mkpp(br,dco);                       % build pp form for derivative
```

下面给出了使用 mmppder 进行分段多项式微分的一个例子：

```
>> x = (0:.1:1)*2*pi;       % same data as earlier
>> y = sin(x);
>> pp = spline(x,y);        % pp-form fitting rough data
>> ppd = mmppder(pp);       % pp-form of derivative
>> xi = linspace(0,2*pi);   % finer points for interpolation
>> yi = ppval(pp,xi);       % evaluate curve
>> yyd = ppval(ppd,xi);     % evaluate derivative
>> plot(x,y,'o',xi,yi,xi,yyd,'--') % plot results
>> title('Figure 21.4: Spline Differentiation')
```

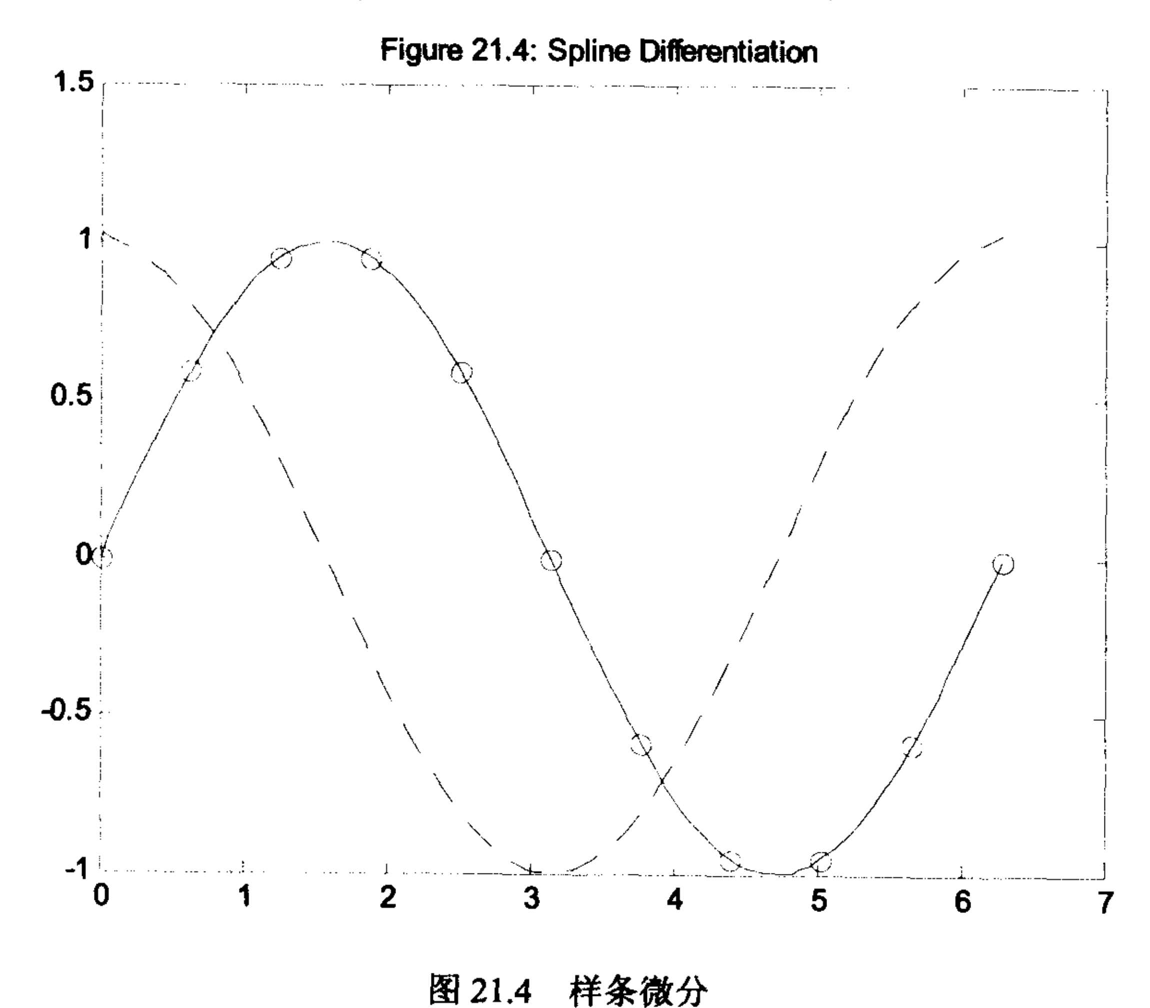


图 21.4　样条微分

说明：图 21.4 证明了下式成立：

$$\frac{\mathrm{d}}{\mathrm{d}x}\sin(x)=\cos(x)$$

21.6　平面上的样条插值

我们在介绍 spline 函数时讲到，要利用该函数实现样条插值必须保证原始数据随自变量单调变化。假如 spline 要利用三次样条插值来描述函数 $y=s(x)$，则该函数的 x 与 y 必须是一一对应的关系，如果函数不是单调的，则函数 ppval 就无法知道对一个给定的 x 应该返

回哪个 y 值（因为此时可能有好几个 y 值与 x 对应）。下面的代码给出了一个非单调的函数的例子，该例在一个平面上画一个螺旋形的曲线：

```
>> t = linspace(0,3*pi,15);
>> x = sqrt(t).*cos(t);
>> y = sqrt(t).*sin(t);

>> plot(x,y)
>> xlabel('X')
>> ylabel('Y')
>> title('Figure 21.5: Spiral Y=f(X)')
```

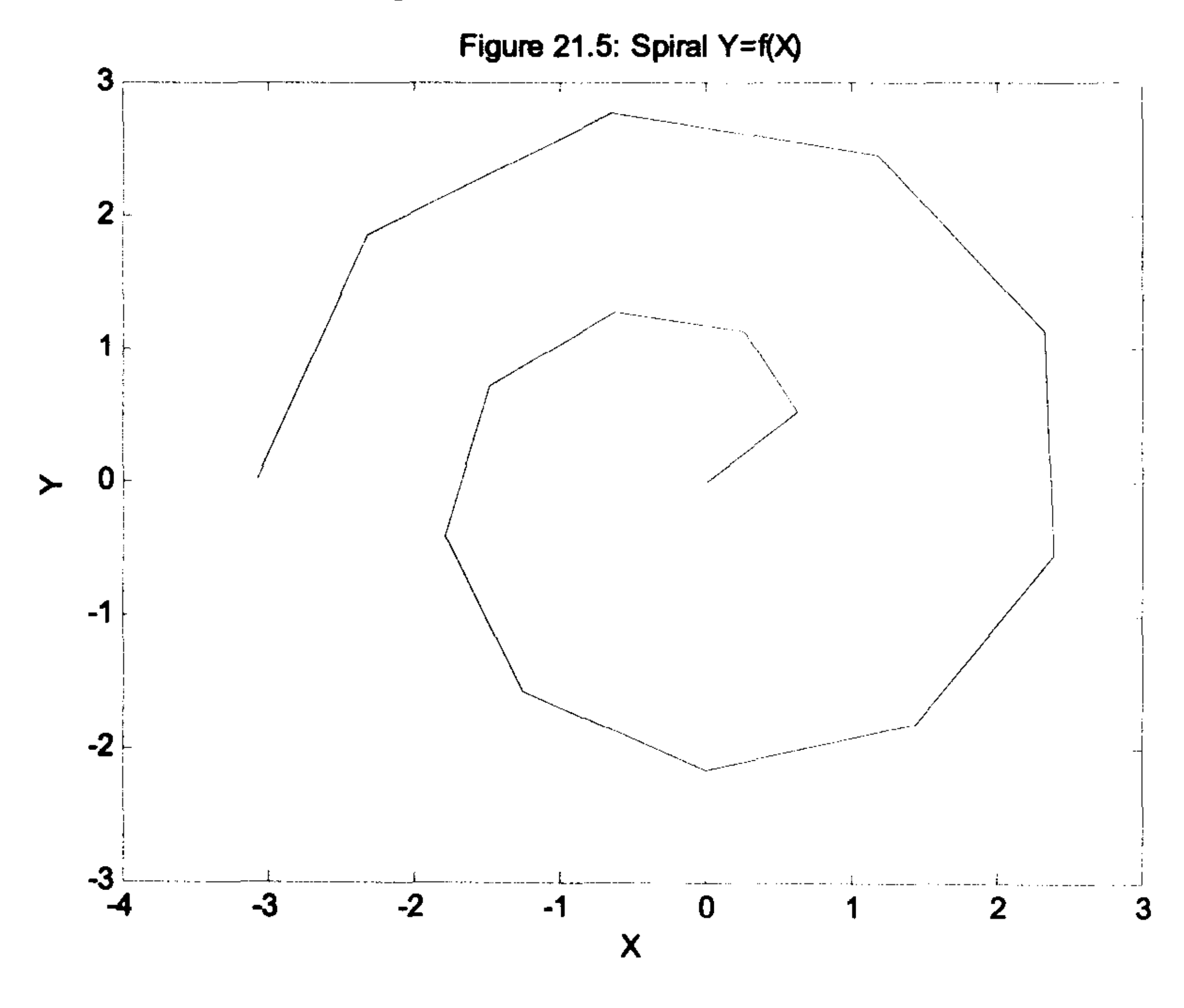


图 21.5 螺旋线 $Y=f(X)$

从图 21.5 可以看出，该曲线不是单调的，因为对一个给定的 x，可能有好几个 y 与之对应。因此，Matlab 就无法给这个螺旋形曲线计算出一个用于插值的三次样条多项式。但是，我们发现，虽然 y 相对于 x 不是单调的，但我们可以将 x 的取值与 y 的取值分别看作是另一个自变量 t 的函数，并使 x 与 y 都相对于 t 单调，这样，我们就可以为每一个轴计算出一个相对于 t 的样条多项式，然后再绘制出螺旋形曲线。在 Matlab 中，可以用两种方法来实现这一操作：第一种方法是，用户可以调用一次 spline 来拟合出一个样条多项式 $x(t)$，然后再调用一次 spline 拟合出 $y(t)$；另一种方法是，spline 函数可以同时拟合出两个样条多项式 $x(t)$和 $y(t)$，然后返回一个包含了两个拟合多项式的 pp 格式结构体。下面的代码给出了第二种实现方法：

```
>> ppxy = spline(t,[x;y])
ppxy =
     form: 'pp'
   breaks: [1×15 double]
```

```
   coefs: [28×4 double]
  pieces: 14
   order: 4
     dim: 2
```

上例中，spline 函数的第二个参数是一个两行的数组，第一行为 x 的取值，第二行为相应的 y 的取值，它们都相对于 t 单调。

注意：函数 spline 与其他的大部分数组处理函数不同，它采用行方向提取输入参数中的数据，而不是采用列方向，也就是说，在 spline 函数中，不同的行表示不同的处理变量。读者如果不了解这一点，很有可能导致错误产生，如下面的代码所示：

```
>> ppz = spline(t,[x;y]')       % try "normal" column-oriented data
??? Error using ==> spline
Abscissa and ordinate vector should be of the same length.
```

另外，从上边返回的 pp 格式结构体中我们发现，ppxy.dim=2，这意味着 ppxy 描述的是一个二维样条多项式。

当我们获得 ppxy 后，就可以利用下面的代码对螺旋曲线函数进行插值操作：

```
>> ti = linspace(0,3*pi);        % total range,100 points
>> xy = ppval(ppxy,ti);          % evaluate both splines
>> size(xy)                      % results are row-oriented too!
ans =
    2       100
>> plot(x,y,'d',xy(1,:),xy(2,:))
>> xlabel('X')
>> ylabel('Y')
>> title('Figure 21.6: Interpolated Spiral Y=f(X)')
```

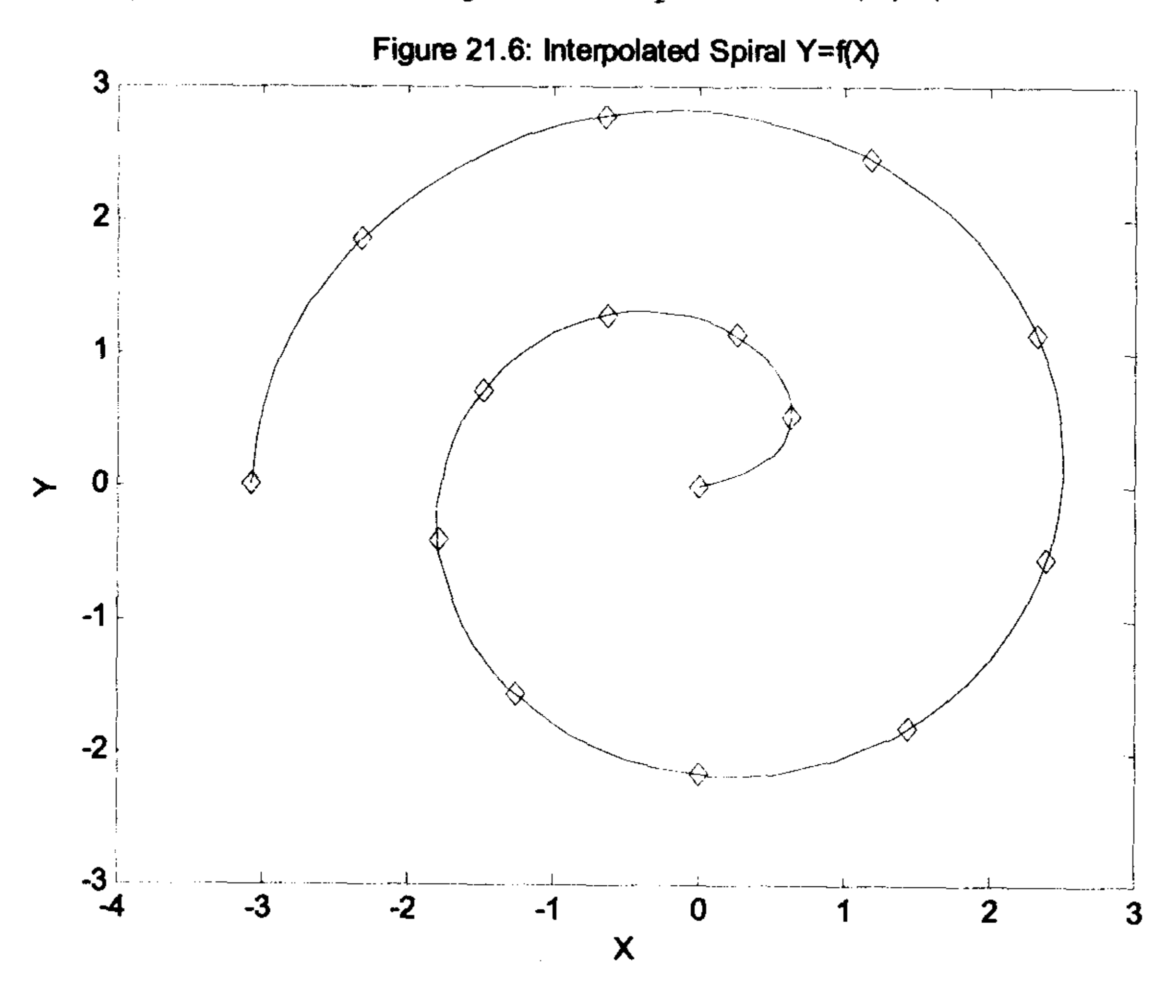


图 21.6　插值后的螺旋线 $Y=f(X)$

上例中，ppval 函数也返回一个两行的数组 xy，第一行与第一个样条多项式（$x(t)$）有关，第二行与第二个样条多项式（$y(t)$）有关。因此，为了绘制出 y 关于 x 的图形，plot 函数将 xy 的第一行 xy(1,:)作为横坐标，将 xy 的第二行 xy(2,:)作为纵坐标进行绘制。

最后需要指出，上边的方法并不只局限于二维情况，所有的 pp 格式结构体和 Matlab 分段多项式函数都可以处理 n 维的样条插值操作。

Chapter 22

傅里叶分析

傅里叶分析是数字信号处理的基础，是频域分析的重要工具，包括连续傅里叶级数、连续傅里叶变换、离散时间傅里叶级数以及离散时间傅里叶变换，这些变换都将一个信号分解成表征信号频域特性的不同正弦波分量的组合。Matlab 提供了函数 fft、ifft、fft2、ifft2、fftn、ifftn、fftshift 和 ifftshift 用于进行傅里叶分析。这些函数能够实现一维或者多维的离散傅里叶变换及其反变换。如果读者的 Matlab 带有数字信号处理工具箱，则可以在那里找到更多的信号处理工具和函数。

信号处理是一个十分广泛的领域，涵盖众多不同的内容，因此本章不是在教会读者如何使用 Matlab 提供的离散傅里叶变换函数解决各种数字信号处理问题（如果读者需要，可以参考一些基于 Matlab 的数字信号处理方面的参考书），而只通过向读者展示如何用函数 fft 来逼近一个连续时间信号和一个周期性的连续时间信号的傅里叶级数，说明离散傅里叶变换函数的基本用法。

22.1 离散傅里叶变换

在 Matlab 中，一个信号的离散傅里叶变换是通过函数 fft 计算的。当信号数据的长度是 2 的 n 次方或是若干个质数的乘积时，fft 函数将采用快速傅里叶变换（FFT）算法来计算离散傅里叶变换。

> 从数字信号处理的角度讲，在数据长度为 2 的 n 次方时，离散傅里叶变换的计算速度会有飞速提高，因此建议用户在任何可能的情况下，都将要进行变换的数据长度变为 2 的 n 次方，如果原始数据长度不够长，用户可以通过补 0 使其长度达到 2 的 n 次方。

Matlab 中使用的 FFT 和通常教科书中使用的 FFT 的形式是一样的，即：

$$F(k) = FFT\ \{f(n)\} = \sum_{n=0}^{N-1} f(n)e^{-j2\pi nk/N} \qquad k = 0,1,\cdots,N-1$$

由于 Matlab 的最小索引值为 1，不支持 0 索引，因此我们需要对上式稍加改动：

$$F(k)=FFT\ \{f(n)\}=\sum_{n=1}^{N}f(n)e^{-j2\pi(n-1)(k-1)/N}\qquad k=1,2,\cdots,N$$

与上式对应的快速傅里叶逆变换（IFFT）为：

$$f(n)=FFT^{-1}\{F(k)\}=\frac{1}{N}\sum_{k=1}^{N}F(k)e^{j2\pi(n-1)(k-1)/N}\qquad n=1,2,\cdots,N$$

fft 函数的用法可以通过 Matlab 提供的该函数的帮助文档获得，如下所示：

```
>> help fft
 FFT Discrete Fourier transform.
   FFT(X) is the discrete Fourier transform (DFT) of vector X.  For
   matrices, the FFT operation is applied to each column. For N-D
   arrays, the FFT operation operates on the first non-singleton
   dimension.

   FFT(X,N) is the N-point FFT, padded with zeros if X has less
   than N points and truncated if it has more.
   FFT(X,[],DIM) or FFT(X,N,DIM) applies the FFT operation across the
   dimension DIM.
   For length N input vector x, the DFT is a length N vector X,
   with elements
                    N
      X(k) =       sum  x(n)*exp(-j*2*pi*(k-1)*(n-1)/N), 1 <= k <= N.
                   n=1
   The inverse DFT (computed by IFFT) is given by
                    N
      x(n) = (1/N) sum  X(k)*exp( j*2*pi*(k-1)*(n-1)/N), 1 <= n <= N.
                   k=1

   See also fft2, fftn, fftshift, fftw, ifft, ifft2, ifftn.
```

为了说明 FFT 的用法，我们来考虑一个简单的例子。假如给定一个信号如下：

$$f(t)=2\mathrm{e}^{-3t}\qquad t\geq 0$$

该信号的连续傅里叶变换为：

$$F(\omega)=\frac{2}{3+j\omega}$$

现在的任务是要用 Matlab 提供的函数逼近上面的连续傅里叶变换 $F(\omega)$。

需要说明的是，在上面的例子中，使用 FFT 来逼近 $F(\omega)$ 几乎没有太大的意义，因为例子中的信号是我们常见的信号，其 $F(\omega)$ 本来就是已知的，但我们这里是向读者展示一种连续傅里叶变换的方法，这对于用户估计一个陌生信号的连续傅里叶变换有很大的帮助。利用傅里叶分析函数估计 $|F(\omega)|$ 的代码如下：

```
N = 128;                 % choose a power of 2 for speed
t = linspace(0,3,N);     % time points for function evaluation
f = 2*exp(-3*t);         % evaluate function, minimize aliasing: f(3) ~ 0
```

```
Ts = t(2) - t(1);         % the sampling period
Ws = 2*pi/Ts;             % the sampling frequency in rad/sec
F = fft(f);               % compute the fft
Fc = fftshift(F)*Ts;      % shift and scale

W = Ws*(-N/2:(N/2)-1)/N; % frequency axis
Fa = 2./(3+j*W);          % analytical Fourier transform
plot(W,abs(Fa),W,abs(Fc),'.') % generate plot, 'o' marks fft
xlabel('Frequency, Rad/s')
ylabel('|F(\omega)|')
title('Figure 22.1:  Fourier Transform Approximation')
```

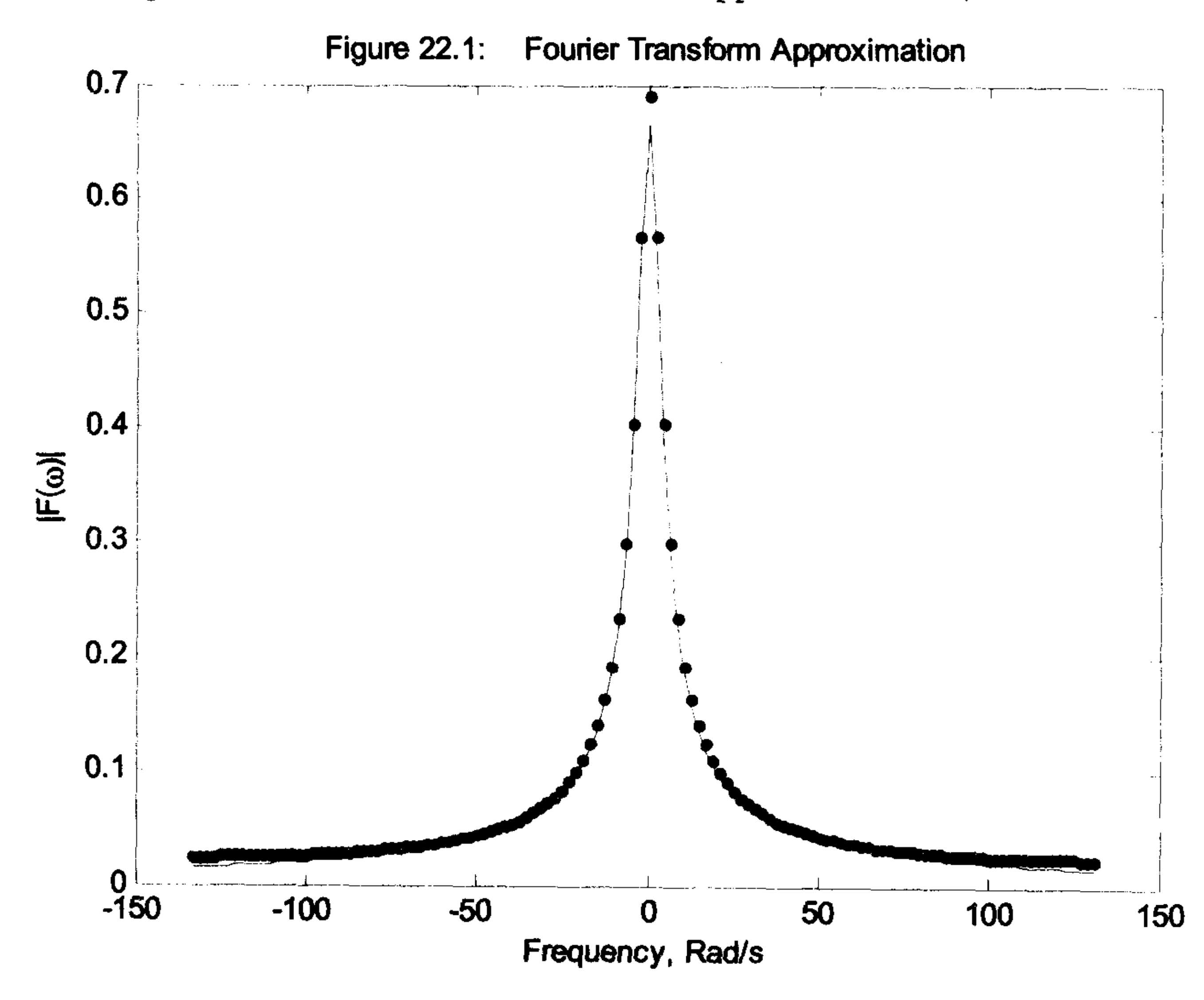


图 22.1 傅里叶变换的估计

图 22.1 给出了估计的连续傅里叶变换和计算出的 $F(\omega)$ 之间的比较。

上例中的函数 fftshift 用于将 F 进行半值翻转，以便使得 Fc 的第(N/2)+1 个元素成为最终结果的直流成分，该元素左边的元素是负频率成分，右边的元素则是正频率成分。W 用于创建频率轴，且 W(N/2+1)处的频率为 0。从图 22.1 可以看出，在低频部分，FFT 近似的效果是很好，但在接近 Nyquist 频率的高频部分却表现出了一定的频率偏差。

Matlab 的其他傅里叶分析函数的用法都与 fft 大同小异，这里不再赘述。这里仅以下面表格的形式给出与 FFT 有关的函数列表：

函数	描述
conv	卷积
conv2	二维卷积

（续表）

函数	描述
convn	n 维卷积
deconv	反卷积
filter	一维数字滤波
filter2	二维数字滤波
fft	离散傅里叶变换
fft2	二维离散傅里叶变换
fftn	n 维离散傅里叶变换
ifft	离散傅里叶逆变换
ifft2	二维离散傅里叶逆变换
ifftn	n 维离散傅里叶逆变换
fftshift	翻转 FFT 的结果以增加负频率轴和谱中心零位
ifftshift	取消 fftshift 执行的操作
abs	求复数数组的幅度
angle	求复数数组的相角
unwrap	矫正相位突变
cplxpair	将数据按共轭复数对重新排序
nextpow2	找出比输入数据大的最近的 2 的 n 次幂

22.2　傅里叶级数

Matlab 没有专门提供函数来进行傅里叶级数分析和处理。不过，如果用户对周期信号采样的离散傅里叶变换和傅里叶级数之间的关系比较熟悉，可以自己编写程序添加这项功能。

一个实周期信号 $f(t)$ 的傅里叶级数可以用下面的复指数形式表示：

$$f(t)=\sum_{n=-\infty}^{\infty}F_n\mathrm{e}^{jn\omega_0 t}$$

其中，傅里叶级数系数由下式给出：

$$F_n=\frac{1}{T_0}\int_t^{t+T_0}f(t)\mathrm{e}^{-jn\omega_0 t}\mathrm{d}t$$

上式中的基本频率（简称基频）为 $\omega_0=2\pi/T_0$，T_0 为信号周期。

傅里叶级数也可以使用三角形式表示如下：

$$f(t)=A_0+\sum_{n=1}^{\infty}\{A_n\cos(n\omega_0 t)+B_n\sin(n\omega_0 t)\}$$

上式中的各个系数分别为：

$$A_0 = \frac{1}{T_0}\int_t^{t+T_0} f(t)\mathrm{d}t$$

$$A_n = \frac{2}{T_0}\int_t^{t+T_0} f(t)\cos(n\omega_0 t)\mathrm{d}t$$

$$B_n = \frac{2}{T_0}\int_t^{t+T_0} f(t)\sin(n\omega_0 t)\mathrm{d}t$$

在傅里叶级数的两种表示形式中，复指数形式便于进行分析和处理，三角形式则便于用户直观理解，因为它使用户很容易看到其中的正弦和余弦信号。另外，这两种表达形式可以相互转换，其系数之间的转换关系是：

$$A_0 = F_0$$

$$A_n = 2\,\mathrm{Re}\{F_n\}$$

$$B_n = -2\,\mathrm{Im}\{F_n\}$$

$$F_n = F^*{}_{-n} = (A_n - jB_n)/2$$

利用上述转换关系，用户可以根据需要在计算时选择任何一种表达方式，然后再将结果进行转换就可以获得另一种表达方式。

当用户选择好时间采样点后，就可以利用离散傅里叶变换计算傅里叶级数的系数，但注意需要将变换的输出进行换算。例如，假如要计算图 22.2 中的锯齿波的傅里叶级数系数。

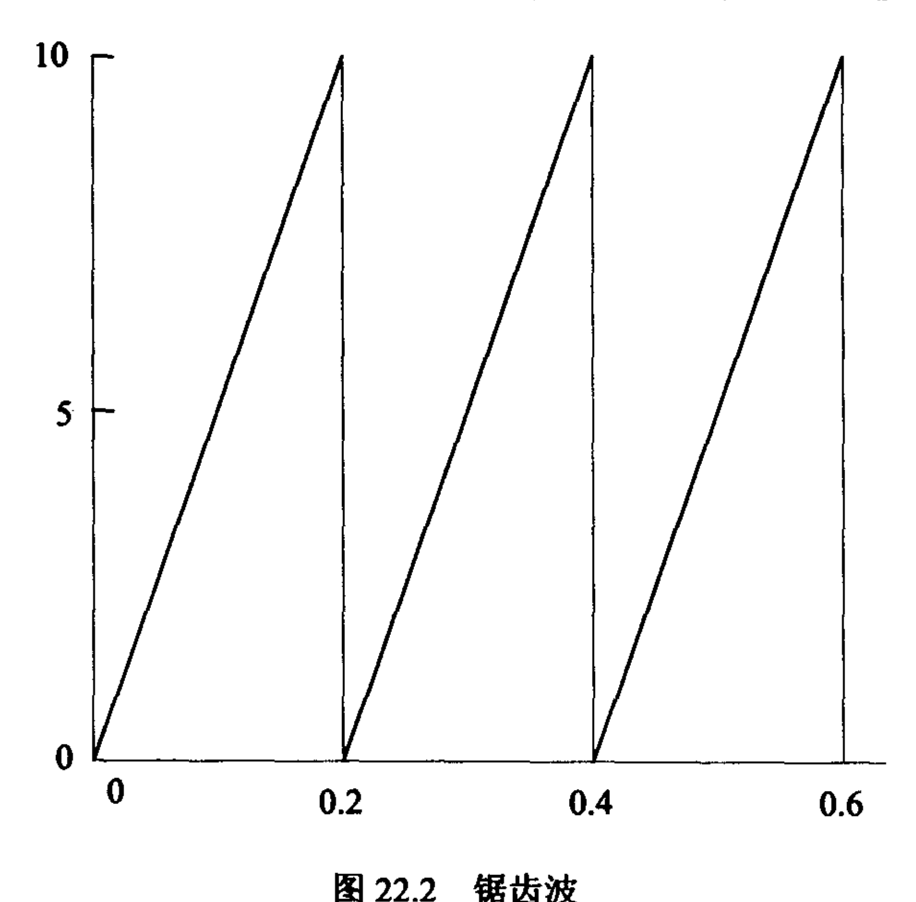


图 22.2 锯齿波

上面的锯齿波函数可以通过下面用户创建的 M 文件函数来产生：

```
function f=sawtooth(t,To)
%SAWTOOTH Sawtooth Waveform Generation.
% SAWTOOTH(t,To) computes values of a sawtooth having
% a period To at the points defined in the vector t.
f = 10*rem(t,To)/To;
f(f==0 | f==10) = 5; % must average value at discontinuity!
```

在进行傅里叶变换时，为了减小偏差，必须选择足够多的谐波，但又不能太多，一般情况下，对谐波的选择视用户的要求而定，只要用户觉得谐波的幅度在计算时不能忽略，都必须将这些谐波包括在内。本例中，我们选择 25 个谐波来计算：

```
>> N = 25;          % number of harmonics
>> To = 0.2;        % choose period
```

由于离散傅里叶变换分别要计算正谐波和负谐波，所以实际需要计算的周期数是谐波的两倍，即：

```
>> n = 2*N;
```

根据奈奎斯特准则，函数必须在一个周期内计算 n 个点的值，因此，在取点时要使第(n+1)个点与第一个点相距一个周期的距离，即：

```
>> t =linspace(0,To,n+1);      % (n+1)th point is one period away
>> t(end) = [];                % throw away undesired last point
>> f = sawtooth(t,To);         % compute sawtooth
```

获得了锯齿波数据后，就可以计算其傅里叶变换了：

```
>> Fn = fft(f);                                    % compute FFT
```

将 Fn 进行重新排列，就可以得到复指数形式的傅里叶级数系数：

```
>> Fn = [conj(Fn(N+1)) Fn(N+2:end) Fn(1:N+1)];    % rearrange values
>> Fn = Fn/n;                                      % scale results
```

现在，变量 Fn 中包含了升序排列的复指数形式的傅里叶级数系数。也就是说，Fn(1)是 F_{-25}，Fn(26)是 F_0，而 Fn(51)是 F_{25}。

我们也可以利用下面的代码将 Fn 转换为三角形式的傅里叶级数系数：

```
>> A0 = Fn(N+1) % DC component
A0 =
     5
>> An = 2*real(Fn(N+2:end)) % Cosine terms
An =
   1.0e-015 *
   Columns 1 through 7
     -0.1176    -0.0439   -0.2555   0.3814    0.0507    -0.2006     0.1592
   Columns 8 through 14
     -0.1817     0.0034    0         0.0034   -0.1141    -0.1430    -0.0894
   Columns 15 through 21
     -0.0685    -0.0216    0.0537 -0.0496    -0.0165    0          -0.0165
```

```
  Columns 22 through 25
   -0.0079    0.2405    0.3274    0.2132
>> Bn = -2*imag(Fn(N+2:end)) % Sine terms
Bn =
  Columns 1 through 7
   -3.1789   -1.5832   -1.0484   -0.7789   -0.6155   -0.5051   -0.4250
  Columns 8 through 14
   -0.3638   -0.3151   -0.2753   -0.2418   -0.2130   -0.1878   -0.1655
  Columns 15 through 21
   -0.1453  -0.1269   -0.1100   -0.0941   -0.0792   -0.0650   -0.0514
  Columns 22 through 25
   -0.0382  -0.0253  -0.0126         0
```

由于锯齿波波形是奇对称的，因此其傅里叶级数系数中的余弦成分 An 应为 0，但在 Matlab 计算是，这些值并不是绝对为零的，而是一系列极小的数（10^{-15} 数量级）。

下面的代码将锯齿波的实际傅里叶级数系数和上边计算的 Bn 进行比较，得到了它们之间的相对误差：

```
>> idx = -N:N;           % harmonic indices
>> Fna = 5j./(idx*pi);  % complex exponential terms
>> Fna(N+1) = 5;
>> Bna = -2*imag(Fna(N+2:end));     % sine terms
>> Bn_error = (Bn-Bna)./Bna         % relative error
Bn_error =
  Columns 1 through 7
   -0.0013   -0.0053   -0.0119   -0.0211   -0.0331   -0.0478   -0.0653
  Columns 8 through 14
   -0.0857   -0.1089   -0.1352   -0.1645   -0.1971   -0.2330   -0.2723
  Columns 15 through 21
   -0.3152   -0.3620   -0.4128   -0.4678   -0.5273   -0.5917   -0.6612
  Columns 22 through 25
   -0.7363   -0.8174   -0.9051   -1.0000
```

从上面的结果可以看出，随着频率的增大，计算偏差越来越明显，有时甚至会导致错误。这是因为我们在计算时只选择了有限的高频谐波分量，而实际的连续波形都包含无限多的高频谐波分量。因此，由于 Matlab 计算的有限性，计算偏差是无法避免的，但我们可以将其控制在一个能够接受的范围之内。一般而言，在计算时增加谐波的数量，可以从某种程度上减少偏差。因此，为了将偏差减小到最低限度，用户可以先利用较多数量的谐波进行计算，然后选择其中的一部分来进行验证和进一步处理。

在许多数字信号处理文献中，信号频谱很多时候都是用棒棒图给出的，在这里，我们也可以使用 stem 函数画出锯齿波的复指数傅里叶级数的频谱：

```
>> stem(idx,abs(Fn))
>> xlabel('Harmonic Index')
>> title('Figure 22.3: Sawtooth Harmonic Content')
>> axis tight
```

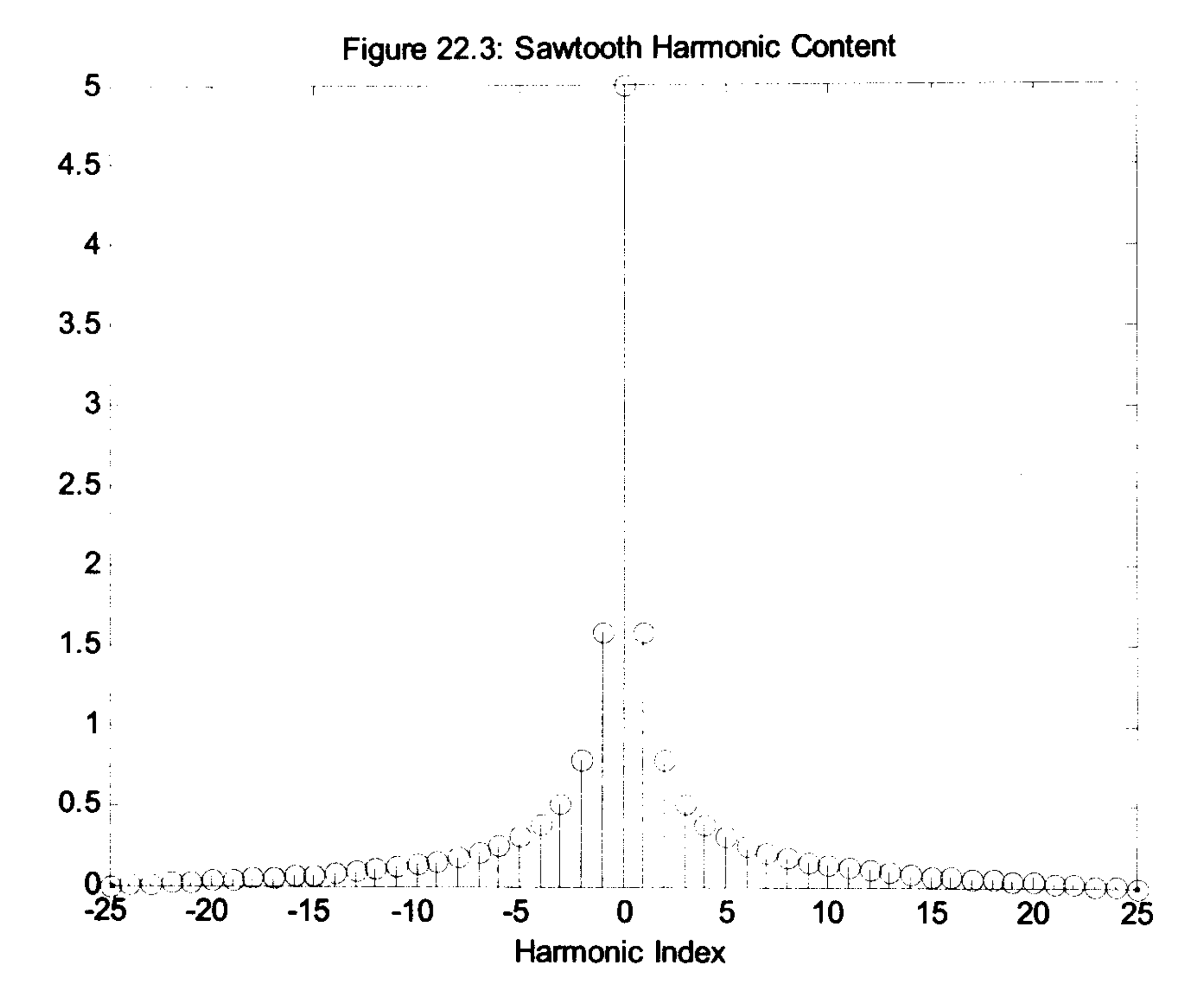


图 22.3 锯齿波谐波频谱图

Chapter 23

优　　化

本章中的优化指的是确定一个函数 $y=g(x)$在什么地方能够获得某一特定值或极值的过程。对于一个简单定义的函数 $y=g(x)$，通常可以找到相应的逆函数 $x=g^{-1}(y)$，以便计算对于给定的 y 值，相应的 x 值的大小。但也有很多函数（包括我们常见的一些函数，如 sin 函数）并不存在严格意义上的逆函数。这时，用户必须通过一个迭代过程来估计给定 y 值时的 x 值。在实际操作过程中，这一迭代过程通常称为寻 0 过程，这是因为对于给定的 y，寻找 x 使 $y=g(x)$，相当于寻找一个 x 值使得 $y-g(x)=0$。

优化的另一项内容是寻找函数的极值，也就是说，找出这个函数在什么地方获得最大或最小值。在很多情况下，函数的极值都必须通过数个周期的循环过程才能估计出来。我们知道，一个函数的最大值其实就是这个函数取负之后的最小值，也就是说，$\max f(x)=\min\{-f(x)\}$，因此，我们在寻找一个函数的极值时，往往只通过循环过程寻找其最小值，这个过程通常被称为最小值算法。

本章介绍了 Matlab 提供的一些基本优化函数。如果读者的 Matlab 带有优化工具箱，可以在那里找到更多的优化工具与函数。

23.1　函数寻零

根据函数不同，可以采用多种方法来求一个函数的零值（即函数等于零时自变量的值）。当函数是一维函数时，可以使用函数 fzero 来寻找其零值。该函数所用到的算法是平分算法和逆二次插值算法的结合算法。当函数是一个多维函数时，也就是说，函数定义是由一个自变量向量的多个标量函数构成，这时，用户无法直接使用基本的优化函数，必须另找解决办法，一个比较好的选择是利用 Matlab 优化工具箱或第三方提供的工具箱来解决。

在后面的章节中，我们将针对 humps 函数演示 Matlab 基本优化函数的用法。下面的代码给出了 humps 函数的波形：

```
>> x = linspace(-.5,1.5);
>> y = humps(x);
>> plot(x,y)
>> grid on
>> title('Figure 23.1: Humps Function')
```

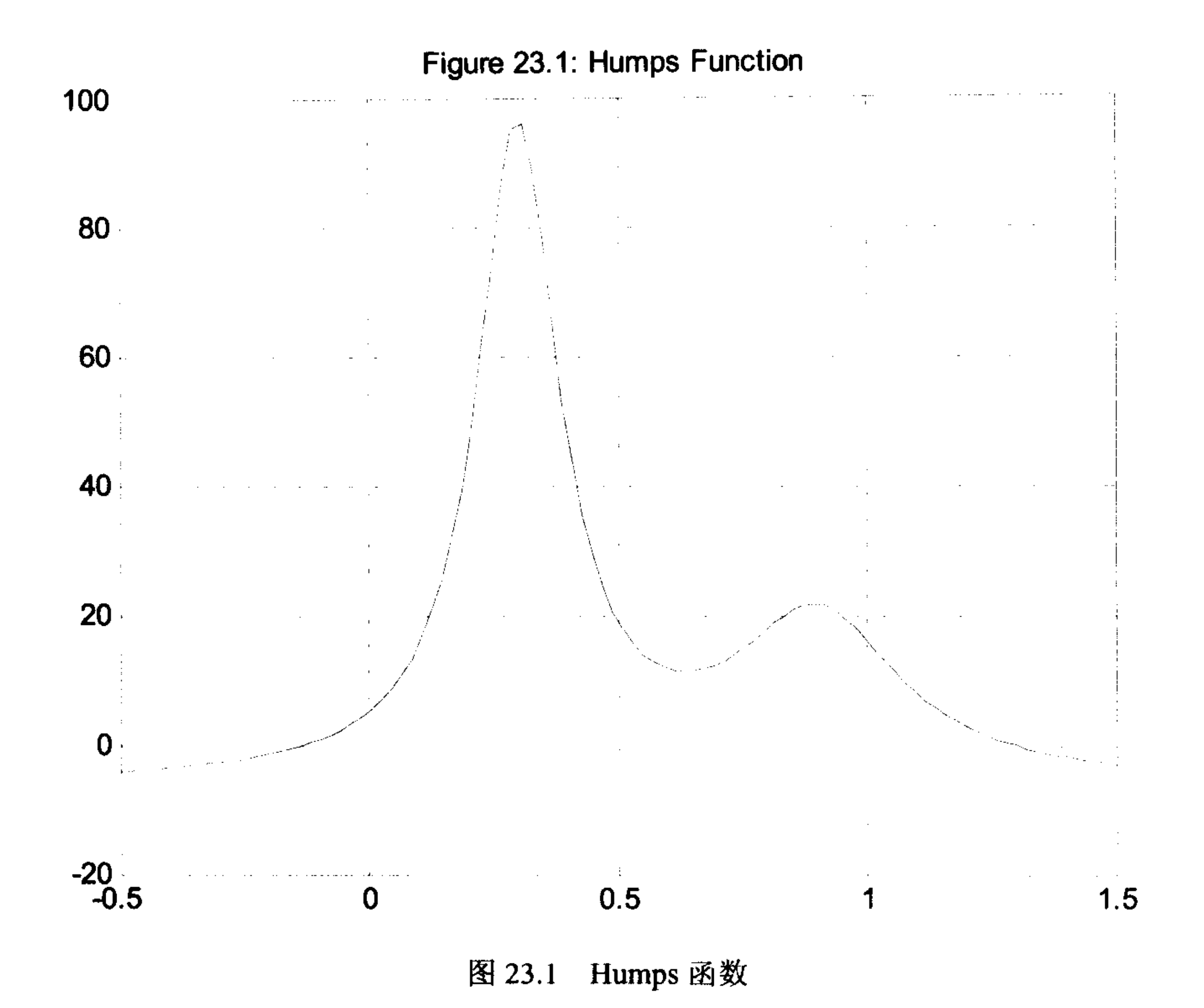


图 23.1 Humps 函数

humps 函数的数学表达式为：

$$humps(x) = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.04} - 6$$

由图 23.1 可知，humps 函数在大约 x=−0.2 和 x=1.3 的地方为 0。

下面的代码利用函数 fzero 分别找出了 humps 函数在 x=1.3 和 x=−0.2 附近的零点位置：

```
>> format long % display more precision

>> H_humps = @humps; % create function handle to humps.m function.

>> x = fzero(H_humps,1.3)
x =
     1.29954968258482
>> humps(x) % how close is it to 0?
ans =
     0
>> H_humps(x) % evaluate humps through its handle as well
ans =
     0
>> [x,value] = fzero(H_humps,-0.2)
x =
     -0.13161801809961
value =
      8.881784197001252e-016
```

上面的代码给出了 fzero 函数的两种调用方式：当只有一个输出参数时，fzero 将返回

估计出的零点，如 x = fzero(H_humps,1.3)，这时用户如果需要验证该零点与实际零点之间的偏差，就需要将这个零点代入原函数进行验证；当有两个输出参数时，fzero 将同时返回估计的零点和相应的偏差值，如[x,value] = fzero(H_humps,-0.2)，因此用户没有必要再调用原函数来检验结果的准确性。有一点值得注意，fzero 每次只返回一个零点——即与用户传递给它的初始估计点最接近的那个零点。因此，如果一个函数有多个零点，用户就需要利用不同的初始估计反复调用 fzero 函数。

如果用户给函数 fzero 一个初始估计，则该函数在搜索零点时，首先在初始估计值两侧搜索函数符号的变化，当发现符号变化时，产生符号变化的这两个值（断点）之间的区域就被锁定为零点搜索区间。这一点很容易理解，因为如果函数是连续的，那么函数必定在出现符号变化的区间内的某个地方过零。锁定搜索区间后，函数 fzero 就利用循环比较搜索最有可能的过零点。

很多时候，用户只知道在什么区间搜索过零点，而不知道过零的大概位置。这时，用户可以将这个初始区间提供给 fzero 函数，而不用提供初始估计。例如，下面的代码利用初始区间估计 humps 的过零点：

```
>> [x,value] = fzero(H_humps,[-2 0])
Zero find in the interval: [-2,0].
x =
    -0.13161801809961
value =
    0

>> [x,value] = fzero(H_humps,[0 1.2])
??? Error using ==> fzero
The function values at the interval endpoints must differ in sign.
```

在第一条语句中，[-2 0]是一个有效的过零区间，因此，fzero 能够找到零点。而第二条语句中，[0 1.2]不是有效的过零区间（因为该区间不存在函数值的符号变化），因此，fzero 无法找到零点并报告一条错误信息。所以，如果用户采用初始区间寻找函数的零点，就必须保证该区间至少包含一个零点，否则 fzero 函数将不会进行零点搜索而直接终止程序。

在上边的例子中，函数都是以函数句柄的形式提供给 fzero 的。第 12 章讲到，在一个函数中调用另一个函数可以采用函数句柄、匿名函数、内联函数和字符串表达式等多种方式。不过在这些方式中，建议读者使用函数句柄的方式。内联函数和字符串表达式方式是老版本使用的方法（不过 Matlab 仍提供对它们的支持），匿名函数则无法用于表示复杂的函数表达式。

需要说明的是，本章中所有的函数（包括优化工具箱中的函数）都有各种可以设置的参数。Matlab 利用相同的格式来管理这些函数的参数，并且提供了两个通用函数 optimset 和 optimget 用于设置和获取函数参数。例如，fzero 有两个可以设置的参数——'Display'和'TolX'。第一个参数用来控制函数工作时返回的细节的数量；第二个参数为接受的最终结果设置一个误差容许范围。下面的代码以 fzero 函数为例向读者展示了函数参数的应用：

```
>> options = optimset('Display','iter'); % show iteration history
>> [x,value] = fzero(H_humps,[-2 0],options)
   Func-count          x                 f(x)              Procedure
```

```
       1              -2          -5.69298     initial
       2               0           5.17647     initial
       3       -0.952481          -5.07853     interpolation
       4       -0.480789          -3.87242     interpolation
       5       -0.240394          -1.94304     bisection
       6       -0.120197           0.28528     bisection
       7       -0.135585        -0.0944316     interpolation
       8       -0.131759       -0.00338409     interpolation
       9       -0.131618       1.63632e-006    interpolation
      10       -0.131618      -7.14819e-010    interpolation
      11       -0.131618                 0     interpolation
Zero found in the interval: [-2, 0].
x =
   -0.13161801809961
value =
       0
>> options = optimset('Display','final'); % display successful interval
>> [x,value] = fzero(H_humps,[-2 0],options)
Zero found in the interval: [-2, 0].
x =
   -0.13161801809961
value =
       0

>> options = optimset('TolX',0.1);
>> [x,value] = fzero(H_humps,[-2 0],options)
x=
   -0.24039447250762
value =
   -1.94303825972565

>> options = optimset('Display','iter','TolX',0.1); % set both
>> [x,value] = fzero(H_humps,[-2 0],options)
   Func-count      x           f(x)          Procedure
       1          -2       -5.69298          initial
       2           0        5.17647          initial
       3   -0.952481       -5.07853          interpolation
       4   -0.480789       -3.87242          interpolation
       5   -0.240394       -1.94304          bisection
Zero found in the interval: [-2, 0].
x =
    -0.24039447250762
value =
    -1.94303825972565
```

前面的例子首先生成一个变量 options 用于保存用户所做的参数设置，该变量只能由 optimset 和 optimget 函数的返回值获得。当用户需要改变 fzero 的参数时，需要首先利用 optimset 函数将期望的参数设置保存到 options 变量中，然后再将此变量作为 fzero 的第三个参数传递给它。对 fzero 而言，'Display'选项包括 4 个设置：'final'、'iter'、'notify'和'off'，其中'notify'是默认设置；'TolX'选项可以是任何有效的数值，其默认值为 eps。关于 Matlab

的优化函数参数的更详细信息，请读者参看 optimset 和 optimget 函数的联机帮助文档。

23.2 一维最小值

除了函数的零点外，用户通常还对函数的极值（最大值（波峰值）和最小值（波谷值））感兴趣。从数学角度来说，极值是函数在其导数（斜率）为 0 的位置的值。例如，上节讲的 humps 函数其极值就出现在导数为零的位置。很明显，对于一个简单的函数，利用导数求极值是方便可行的。但有些函数很难求出其微分函数，有时即使求出微分函数，也很难寻找导数为 0 的点。在这些情况下，就只能通过区域搜索的方法来寻找函数极值了。Matlab 提供了两个函数来完成这项任务：fminbnd 和 fminsearch。这两个函数分别求取一维和 n 维函数的最小值。其中，fminbnd 使用黄金分割和抛物线插值的结合算法来求最小值。由于 $f(x)$的最大值即是$-f(x)$的最小值，因此 fminbnd 和 fminsearch 同样也可以用来求解一个函数的最大值。

我们仍以上节的 humps 函数为例说明如何求取一维函数的最大值和最小值。从图 23.1 中可以看出，曲线在 x=0.3 附近出现了一个最大值，在 x=0.6 附近出现了一个最小值。我们可以利用下面的代码估计 x=0.6 附近的极小值：

```
>> H_humps=@humps;      % create handle to humps.m function.
>> [xmin,value] = fminbnd(H_humps,0.5,0.8)
xmin =
    0.63700821196362
value =
    11.25275412587769
>> options=optimset('Display','iter');
>> [xmin,value] = fminbnd(H_humps,0.5,0.8,options)
Func-count          x              f(x)          Procedure
1               0.61459          11.4103         initial
2               0.68541          11.9288         golden
3               0.57082          12.7389         golden
4               0.638866         11.2538         parabolic
5               0.637626         11.2529         parabolic
6               0.637046         11.2528         parabolic
7               0.637008         11.2528         parabolic
8               0.636975         11.2528         parabolic
Optimization terminated successfully:
the current x satisfies the termination criteria using
OPTIONS.TolX of 1.000000e-004
xmin =
    0.63700821196362
value =
    11.25275412587769
```

在上边的代码中，为了估计 x=0.6 附近的极值，我们在 fminbnd 调用时设定 0.5~0.8 作为搜索范围。另外，在 fminbnd 的第二次调用时，我们对选项进行了设置，以便显示 fminbnd 的搜索过程。

要估计 x=0.3 附近的最大值，我们既可以对 humps.m 文件进行修改，使其符号相反，也可以创建一个与 humps 符号相反的匿名函数。下面给出了采用匿名函数法估计 x=0.3 附近的最大值的实现代码：

```
>> AH_humps = @(x) -1./((x-.3).^2+.01)-1./((x-.9).^2+.04)+6;
>> [xmax,value] = fminbnd(AH_humps,0.2,0.4,options)

 Func-count          x            f(x)             Procedure
     1          0.276393      -91.053             initial
     2          0.323607     -91.4079             golden
     3          0.352786     -75.1541             golden
     4          0.300509     -96.5012             parabolic
     5          0.300397     -96.5014             parabolic
     6          0.300364     -96.5014             parabolic
     7          0.300331     -96.5014             parabolic
Optimization terminated successfully:
the current x satisfies the termination criteria using
OPTIONS.TolX of 1.000000e-004
xmax =
    0.30036413790024
value =
  -96.50140724387050
```

注意，上例返回的极值与实际的极值符号正好相反，因为 fminbnd 估计的是-humps(x)的最小值，也就是 humps 的最大值。从上述结果可以看出，humps 的峰值位置非常接近于 0.3，且峰值幅度约为 96.5。

23.3 多维最小值

在 Matlab 中，函数 fminsearch 可用于计算一个多变量函数 $f(x)$的最小值。其中，x 为一个由多个自变量构成的向量，$f(x)$则是关于 x 的标量函数（即该函数的各个系数参数都是标量）。fminsearch 函数内部应用了 Nelder-Mead 单一搜索算法，通过调整 x 的各个元素的值来寻找 $f(x)$的最小值。该算法虽然对于平滑函数搜索效率没有其他算法高，但它不需要梯度信息，从而使其应用范围大大扩展。因此，该算法特别适用于不太平滑、难以计算梯度信息或梯度信息价值不大的函数。

下面我们通过一个例子来阐述 fminsearch 的用法。考虑下边这个“香蕉”函数（也称为 Rosenbrock 函数）：

$$f(x)=100(x_2-x_1^2)^2+(1-x_1)^2$$

下面的 M 脚本文件画出了上述函数的三维图形：

```
x = [-1.5:0.125:1.5];              % range for x1 variable
y = [-.6:0.125:2.8];               % range for x2 variable

[X,Y] = meshgrid(x,y);             % grid of all x and y
Z = 100.*(Y-X.*X).^2 + (1-X).^2;   % evaluate banana
```

```
mesh(X,Y,Z)
hidden off
xlabel('x(1)')
ylabel('x(2)')
title('Figure 23.2: Banana Function')

hold on
plot3(1,1,1,'k.','markersize',30)
hold off
```

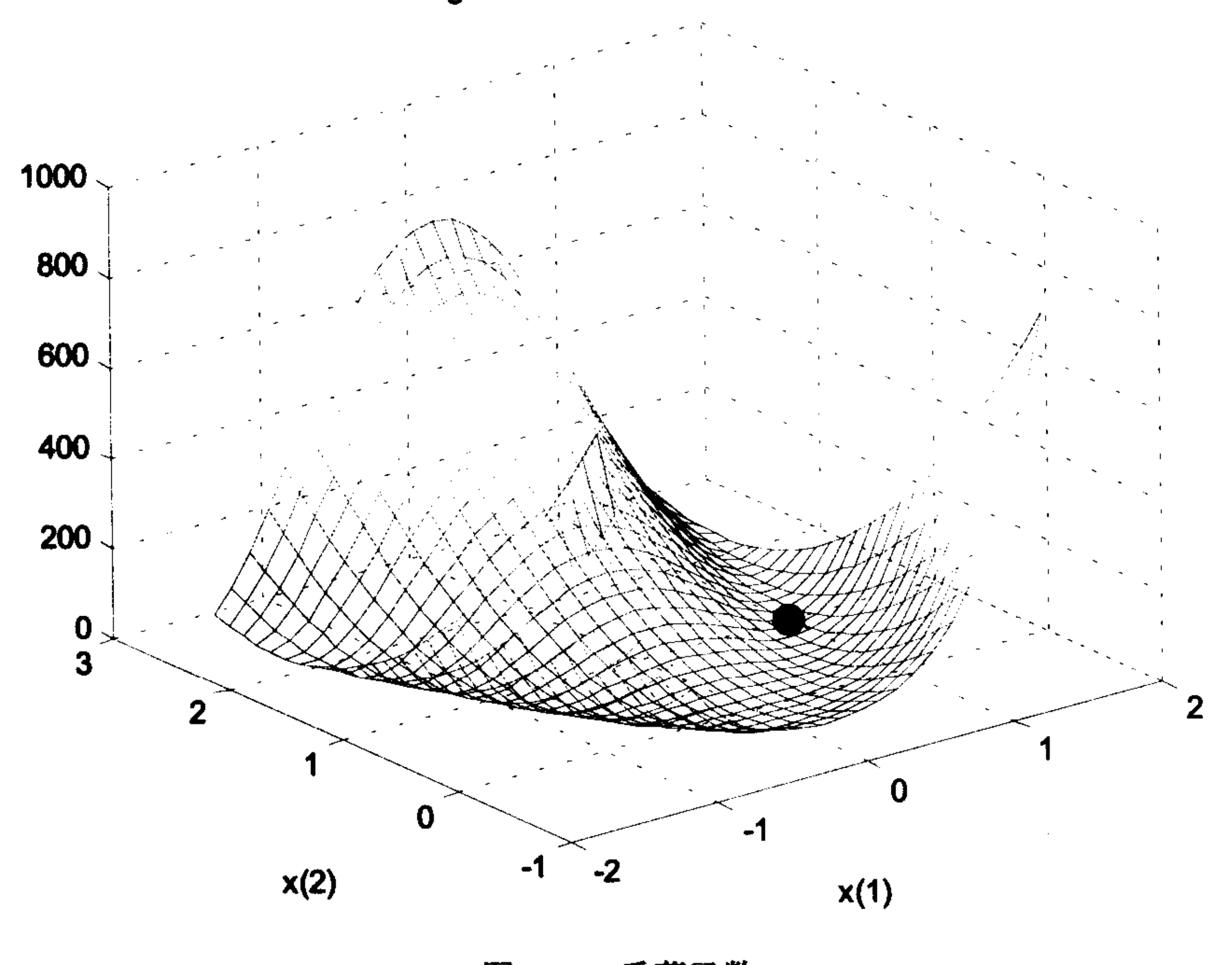


图 23.2 香蕉函数

如图 23.2 所示，该“香蕉”函数在 x=[1;1]附近有一个惟一的最小值 0。为了寻找函数的最小值，需要将函数表达式中的 x_1 设为 $x(1)$，x_2 设为 $x(2)$，并创建一个用于搜索的函数文件。该函数文件可以由下面的“香蕉”M 函数文件给出：

```
function f=banana(x)
% Rosenbrock's banana function
     f=100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
```

也可以由下面的匿名函数给出：

```
>> AH_banana = @(x) 100*(x(2)-x(1)^2)^2+(1-x(1))^2;
```

使用上述任何一种函数形式，都可以利用 fminsearch 函数搜索“香蕉”函数的最小值。下面给出了匿名函数搜索最小值的代码：

```
>> [xmin,value,flag,output] = fminsearch(AH_banana,[-1.9,2])
```

```
xmin =
    1.00001666889480       1.00003447386277
value =
    4.068551535063419e-010
flag =
    1
output =
     iterations: 114
      funcCount: 210
      algorithm: 'Nelder-Mead simplex direct search'
        message: [1×196 char]
```

上面的代码返回了 4 个输出参数，其中 xmin 为最小值出现的位置，value 为函数最小值，flag 为运行状态标志符（1 为成功，0 为失败），output 为一个搜索过程统计信息。本例中，output 的值表示此次搜索的误差容限为 1e-4，完成搜索共进行了 114 次循环迭代，先后执行了 210 次香蕉函数。当然，如果用户不需要这么多输出参数，只需要提供感兴趣的参数就可以了。

和 fminbnd 一样，fminsearch 也可以接受一个 options 参数结构。下表给出了 fminsearch 可以设置的选项：

选项名	描述	默认值
'Display'	显示方式，'iter'、'final'、'notify'或者'off'	'notify'（只有当找不到解时才显示信息）
'MaxFunEvals'	最大函数执行次数	200*length(x)
'MaxIter'	最大算法循环次数	200*length(x)
'TolFun'	函数解（极值）的误差容限	1.00E-004
'TolX'	变量解（位置）的误差容限	1.00E-004

下面我们使用上表中的选项在更严格的误差容限下寻找上述问题的解：

```
>> options = optimset('TolFun',1e-8,'TolX',1e-8);
>> [xmin,value,flag,output] = fminsearch(AH_banana,[-1.9,2],options)
xmin =
    1.00000000126077       1.00000000230790
value =
    6.153858843361103e-018
flag =
    1
output =
    iterations: 144
     funcCount: 266
     algorithm: 'Nelder-Mead simplex direct search'
       message: [1×196 char]
```

上例的误差容限要求 fminsearch 求得的极值与实际最小值之间的距离应在 1e-8 之内，而该极值的位置与实际位置之间的差异也不能超过±1e-8。从 output 的值可以看出，随着误差容限的减小，算法迭代的次数和函数执行的次数都明显增加（约增加了 26%）。因此，如果被搜索的函数比较复杂时，就需要在误差容限和迭代次数之间取得一个合理的折衷。

23.4 注意事项

像 fzero、fminbnd 和 fminsearch 这样的函数在利用迭代算法求函数的零点或极值时，都假设这些函数是可以收敛到要搜索的值上的。但实际上，有些函数可能根本不会收敛或者需要迭代多次才能收敛，甚至有时候还会导致 Matlab 错误，这个错误将会中断迭代而不会输出任何结果。为了保证用户能准确有效地求出函数的零点和极值，在使用前面讲到的优化函数时请用户注意以下事项：

（1）最好给函数提供一个良好的初始估计。一个良好的初始估计对本章问题的解决是很重要的，因为这样可以使搜索过程始终处在解的附近，并使找到的结果处在一个最可靠的范围之内。

（2）求多维极值时，如果一个自变量向量中的各元素之间差了好几个数量级，最好将它们换算到同一数量级或相邻的数量级内，以便提高迭代搜索的效率和准确性。例如，如果 $x(1)$为接近 1 的数，而 $x(2)$为接近 1e6 的数，则在函数定义中最好将 $x(2)$除以 1e6，计算完毕后，再将 $x(2)$乘以 1e6 换算回真实的结果。

（3）如果一个问题很复杂，最好将它化整为零，使其简化成一系列变量较少的简单问题进行解决。

（4）要确保优化函数不会返回难以处理的数值，比如 Inf 或 NaN，因为这通常会带来收敛失败。建议用户使用函数 isreal、isfinite 和 isnan 在返回结果之前检测是否返回了这些值。

（5）尽量避免在执行优化函数时使用导致不连续结果的函数，如 abs、min 和 max 等，因为不连续的结果通常会导致收敛失败。

（6）用户可以通过给要进行迭代的函数添加一个限制条件来限制自变量 x 的范围，从而避免迭代算法产生超出范围的值。

Chapter 24

积分和微分

积分和微分是微积分学中的基本工具。从数学角度上讲，积分计算一个函数与自变量轴之间的面积，微分则描述了一个函数的斜率或者梯度。Matlab 提供了一些微积分函数来估计一个函数的积分值和斜率值。这些微积分函数可以估计的函数类型包括：M 文件函数、匿名函数，以及由定义区间内等间隔抽样点列表构成的函数。

24.1 积分

Matlab 提供了 4 个函数来计算函数的积分：quad、quadl、dblquad 和 triplequad。

我们仍用第 23 章给出的 humps 函数来阐述上述积分函数的用法。首先，为了便于分析，我们将第 23 章的图 23.1 拷贝过来，如图 24.1 所示。在图 24.1 中，我们等间隔选取了一些抽样点，并根据这些抽样点画出了一系列梯形。从图 24.1 中可以很明显地看出，梯形面积的和近似等于函数的积分。很明显，随着梯形数量的增加（也就是抽样点增多），函数积分和梯形面积之间的差别越来越小，也就是说，利用梯形面积对积分的估计也就越来越精确。

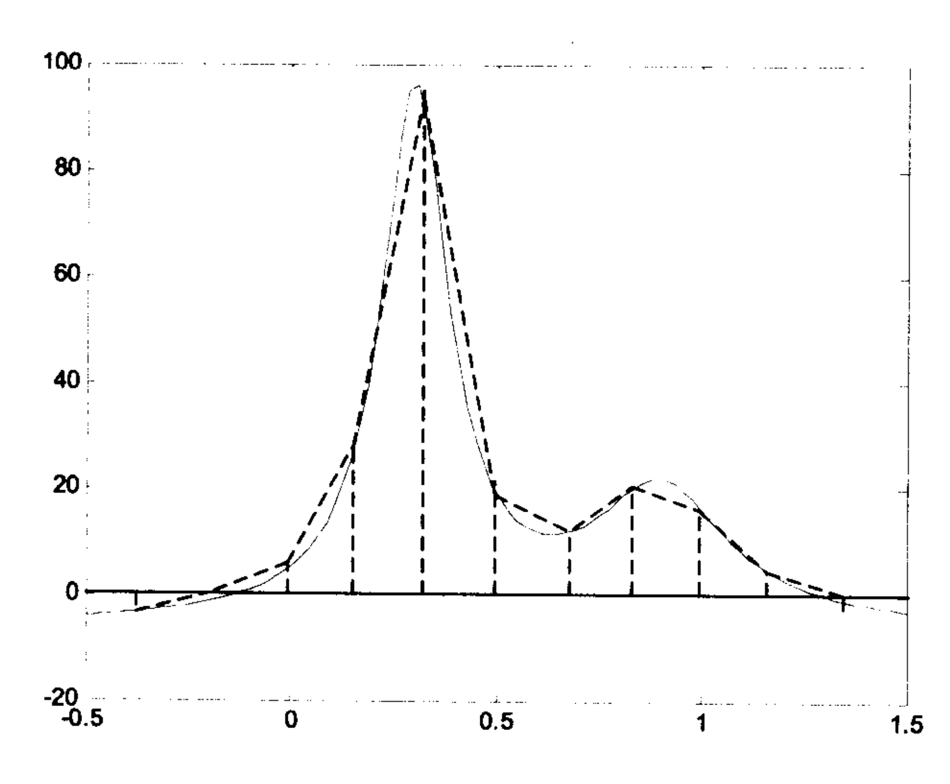


图 24.1 利用梯形分割估计积分

Matlab 函数 trapz 将根据均匀间隔的抽样值列表，使用梯形分割来近似估计函数的面积（积分）。例如，采用图 24.1 中的梯形分割求 humps 函数积分的代码如下：

```
>> x = -1:.17:2;
>> y = humps(x);
>> area = trapz(x,y)
area =
    25.917
```

从图 24.1 可以看出，由于抽样间隔太大，使得这种方法并不能非常精确地估计 humps 函数的积分。不过，用户可以通过缩小抽样间隔，来获得更高的估计精确度。例如，下面的代码采用 100 个抽样点重新估计了 humps 函数的积分：

```
>> x = linspace(-1,2,100);
>> y = humps(x);
>> format long
>> area = trapz(x,y)
area =
    26.34473119524596
```

从结果可以看出，这次估计的精确度远远高于上次。

有时候用户会对下面的积分感兴趣：

$$\int_{x_1}^{x} f(x)\mathrm{d}x$$

其中 x_1 是一个已知的积分下限，x 为自变量。由此可见，该积分是以 x 为自变量的函数。该函数在 x 点的值即是从 x_1 到 x 的有限积分。当利用梯形分割进行累加积分时，可以使用函数 cumtrapz 来计算各个梯形的面积值的一个列表。例如，下面的代码利用 cumtrapz 函数计算用于估计 humps 函数的 100 个梯形的面积值的列表：

```
>> x = linspace(-1,2,100);
>> y = humps(x);
>> z = cumtrapz(x,y);
>> size(z)
ans =
    1       100

>> plotyy (x,y,x,z)
>> grid on
>> xlabel('x')
>> ylabel('humps(x) and integral of humps(x)')
>> title('Figure 24.2: Cumulative Integral of humps(x)')
```

利用我们目前掌握的函数属性，要想确定一个最优的梯形宽度是很困难的。很明显，如果用户能够以某种方式调整各个梯形的宽度，使其符合函数的特点，那么就能够获得较高的估计精确度。

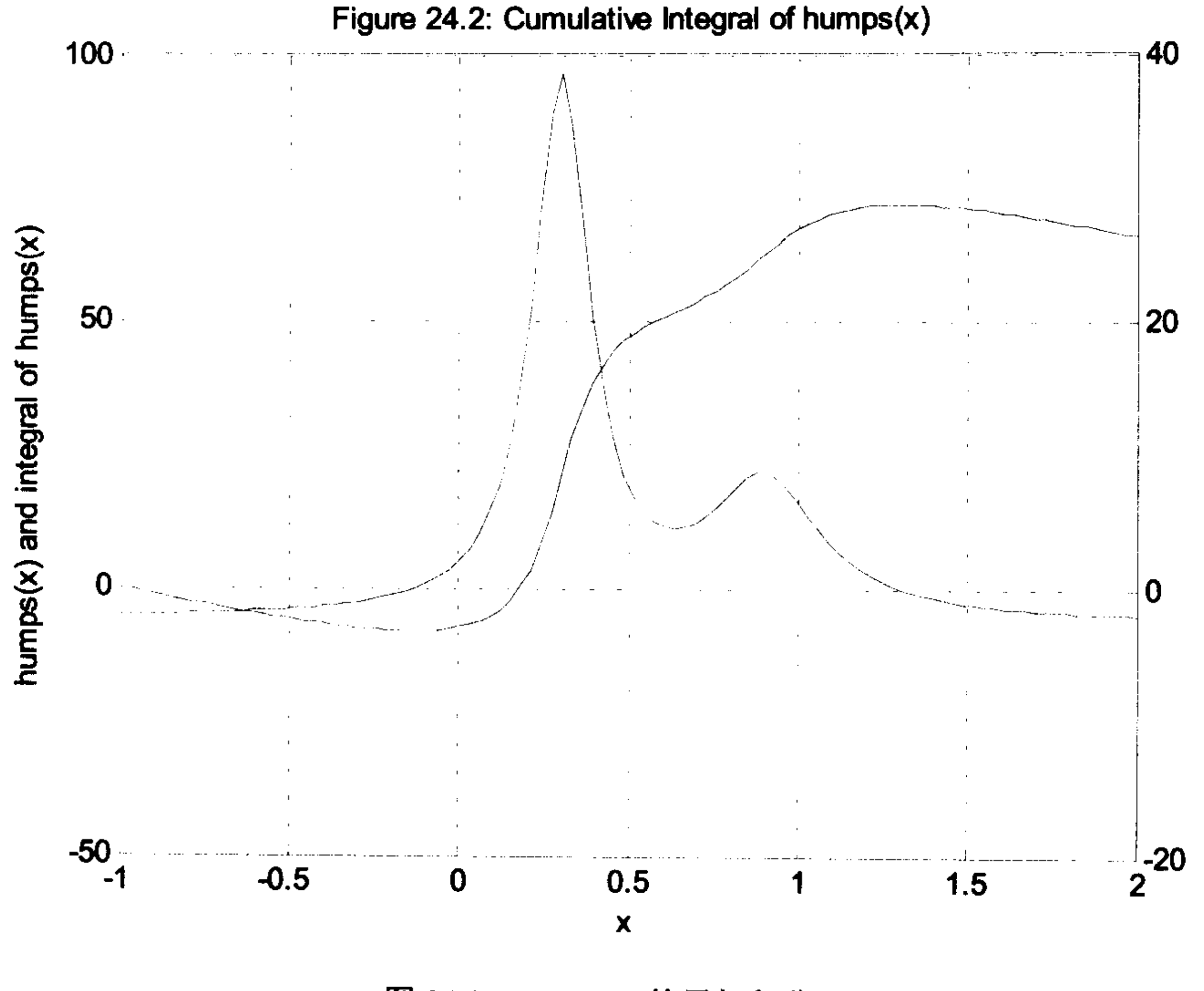


图 24.2　humps(x)的累加积分

Matlab 函数 quad 和 quadl 就是采用上述方法利用 “求面积” 的数学概念估计一个函数的积分值的。这两个积分函数采用几乎相同的方式工作，它们都采用任意间隔的抽样点来提高函数积分值的精确度；另外，这两个函数都对被积函数进行高阶估计，而不是简单的梯形估计，但 quadl 的精确度要比 quad 更高一些。例如，我们可以利用下面的代码再次计算 humps 函数的积分：

```
>> z(end)                        % cumtrapz result
ans =
    26.34473119524596

>> H_humps = @humps;                % Create function handle

>> quad(H_humps,-1,2)
ans =
    26.34496050120123

>> quadl(H_humps,-1,2)
ans =
    26.34496047137897
```

上例中，quad 和 quadl 返回几乎相同的结果，结果表明积分精确度达到了 8 个有效位。读者也可以明显看出 quad 和 quadl 计算出的结果其精确度要高于 cumtrapz 函数（精确度为 5 个有效位）。

进行积分的函数（也叫被积函数）必须支持向量形式的输入，并给出向量形式的输出参数。因此，被积函数大部分都是使用点算术运算符（如 .*、./、.\ 和 .^）来定义的。我们前面使用的 humps 函数就是一个例子，它是采用下面的语句定义的：

```
y = 1./((x-.3).^2+.01)+1./((x-.9).^2 + .04) - 6;
```

quad 和 quadl 函数还允许用户利用其第 4 个输入参数指定一个绝对误差容限。默认的绝对误差容限是 10^{-6}。

除了一维积分之外，Matlab 还提供了二维积分函数：dblquad。dblquad 函数将对如下形式的积分进行估计：

$$\int_{y_{min}}^{y_{max}} \int_{x_{min}}^{x_{max}} f(x,y)\mathrm{d}x\ \mathrm{d}y$$

为了阐明 dblquad 的用法，我们需要首先生成一个二维函数 $f(x,y)$。假如我们使用的二维被积函数是下面的 M 文件函数 myfun：

```
function z=myfun(x,y)
%MYFUN(X,Y) an example function of two variables
z = sin(x).*cos(y) + 1;          % must handle vector x input
```

下面的代码绘制出了该函数的图形：

```
>> x = linspace(0,pi,20);        % xmin to xmax
>> y = linspace(-pi,pi,20);      % ymin to ymax
>> [xx,yy] = meshgrid(x,y);      % create grid of point to evaluate at
>> zz = myfun(xx,yy);            % evaluate at all points
>> mesh(xx,yy,zz)
>> xlabel('x'),ylabel('y')
>> title('Figure 24.3: myfun.m plot')
```

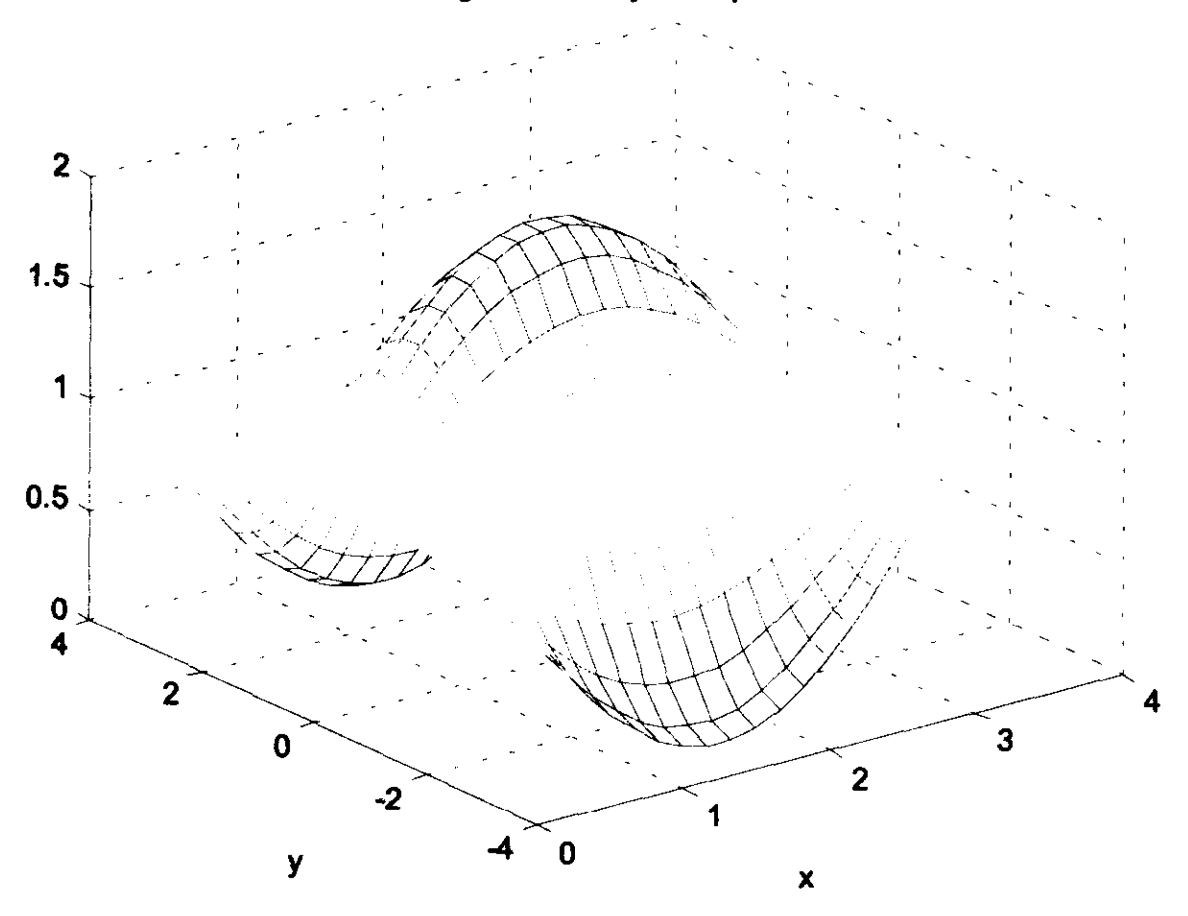


图 24.3 myfun 函数

二维积分实际上计算的是一个曲线与自变量平面之间的体积。下面的代码利用 dblquad 函数计算出了 myfun 曲线与 *x-y* 平面之间的体积：

```
>> area = dblquad(@myfun,0,pi,-pi,pi)
area =
    19.73920880609102
>> relerr = (area-2*pi^2)/(2*pi^2)
relerr =
    1.981996941074027e-010
```

上例中，dblquad 函数的调用方式为 dblquad(Fname,xmin,xmax,ymin,ymax)。实际上，在 dblquad 函数内部仍是调用 quad 函数来计算每一维的积分的，从上面的结果可以看出，dblquad 所得到的结果是相当精确的。

除了前面讲的一维和二维积分外，Matlab 7 还提供了 triplequad 函数进行三维函数的积分估计。triplequad 函数的基本调用方式为 triplequad(Fname,xmin,xmax,ymin,ymax,zmin,zmax)，它计算给定范围内函数 Fname(x,y,z)的积分。有关该函数的更详细信息，请读者参看 Matlab 的帮助文档。

24.2 微分

与积分相比，微分在进行数值计算时要困难得多。如果说积分描述了一个函数的整体或者说是宏观属性，那么微分则描述了一个函数的局部或者说是微观属性。因此，微分不像积分，它对函数波形上的微小变化是非常敏感的，因为函数的微小变化对斜率的影响远远比对面积的影响大。

鉴于微分对数据变化的内在敏感性，用户应尽量避免对数据直接使用数值微分，尤其是当原始数据是通过实验获得时。一种较好的解决办法是先对这些数据进行一次最小二乘曲线拟合，然后再对得到的多项式进行微分；用户也可以先对数据进行三次样条插值拟合，然后再求出其导数的样条插值表达式。为了更形象地说明问题，我们使用 20.8 节中的一个例子来做一些验证，首先需要对这些数据进行曲线拟合，代码如下：

```
>> x = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1];
>> y = [-.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.30 11.2];
>> n = 2; % order of fit
>> p = polyfit(x,y,n)   % find polynomial coefficients
p =
   -9.8108     20.1293     -0.0317

>> xi = linspace(0,1,100);
>> yi = polyval(p,xi); % evaluate polynomial

>> plot(x,y,'-o',xi,yi,'--')
>> xlabel('x'),ylabel('y=f(x)')
>> title('Figure 24.4: Second Order Curve Fitting')
```

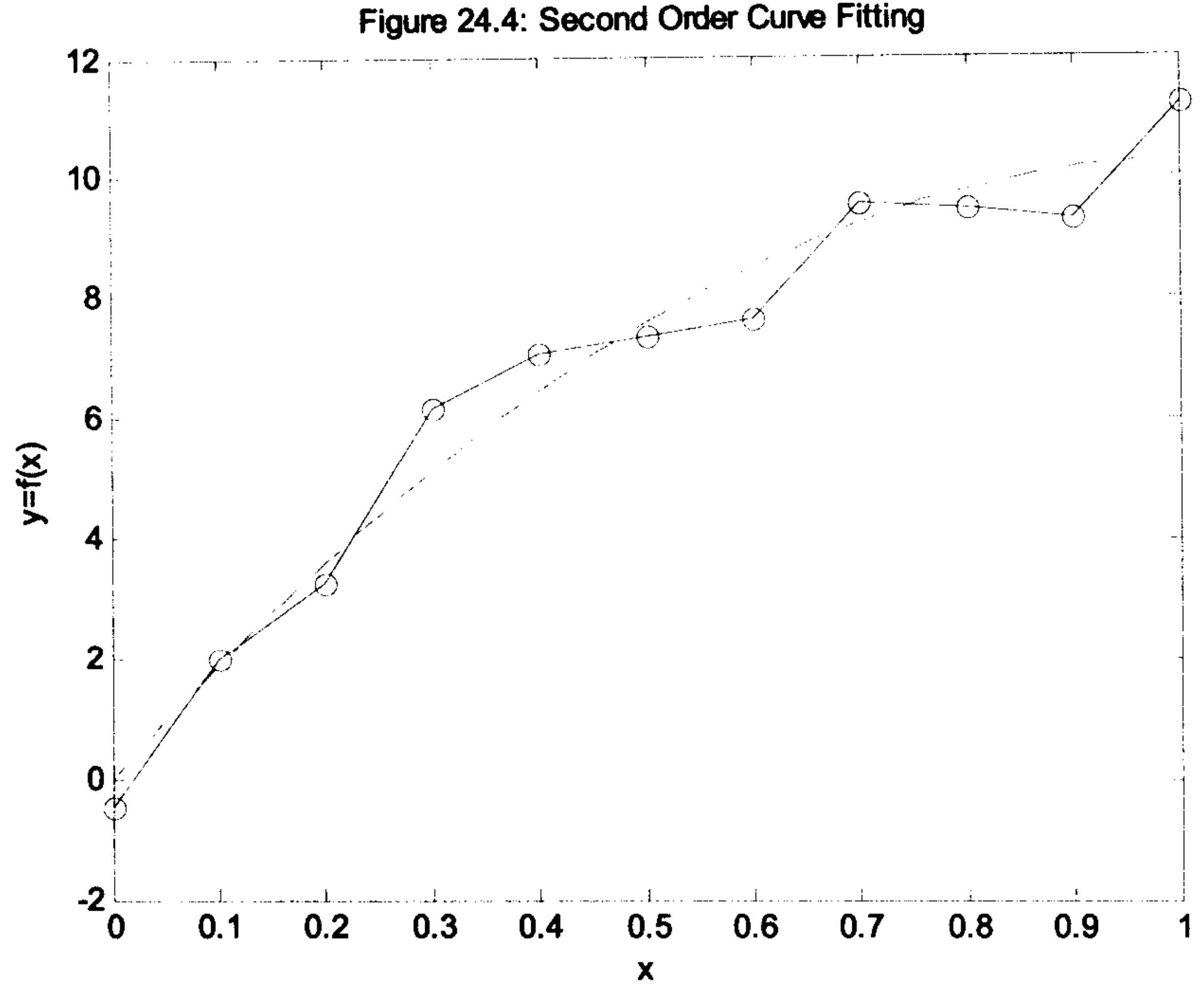


图 24.4　二阶曲线拟合

然后，可以利用求导函数 polyder 求出拟合曲线的导数：

```
>> pd = polyder(p)
pd =
    -19.6217        20.1293
```

从拟合结果看，上例中的拟合曲线应为：$y(x)=-9.8108x^2+20.1293x-0.0317$，因此，其导数为 $dy/dx=-19.6217x+20.1293$。可见，拟合曲线的导数仍然是一个多项式，只不过其阶数为 1，也就是说，该导数是一条直线，它随着 x 的变化做线性变化。

当函数是由一个数据列表描述时，Matlab 也提供了函数 diff 来近似估计该函数的导数。diff 函数实际上是通过计算数组中相邻元素之间的差值来估计导数的。我们知道，导数的定义为：

$$\frac{dy}{dx}=\lim_{\Delta x\to o}\frac{f(x+\Delta x)-f(x)}{\Delta x}$$

在实际应用时，该定义可以用下式近似表示：

$$\frac{dy}{dx}\approx\frac{\Delta y}{\Delta x}=\frac{f(x+\Delta x)-f(x)}{\Delta x}$$

从上式可以看出，导数是通过 y 的前向有限差值（$f(x+\Delta x)-f(x)$）与 x 的前向有限差值（Δx）的比来计算的。而 diff 函数正是用于计算数组元素之间的差值的，因此该函数可以用于近似估计导数。我们仍以前面的数据为例，下面的代码给出了利用 diff 函数直接针对给出的数据进行求导的过程：

```
>> dyp = polyval(dp,x);      % poly derivative for comparison
>> dy = diff(y)./diff(x);    % compute differences and use array division
>> xd =x(1:end-1);           % new x axis array since dy is shorter than y
>> plot(xd,dy,x,dyp,':')
>> ylabel('dy/dx'),xlabel('x')
>> title('Figure 24.5: Forward Difference Derivative Approximation')
```

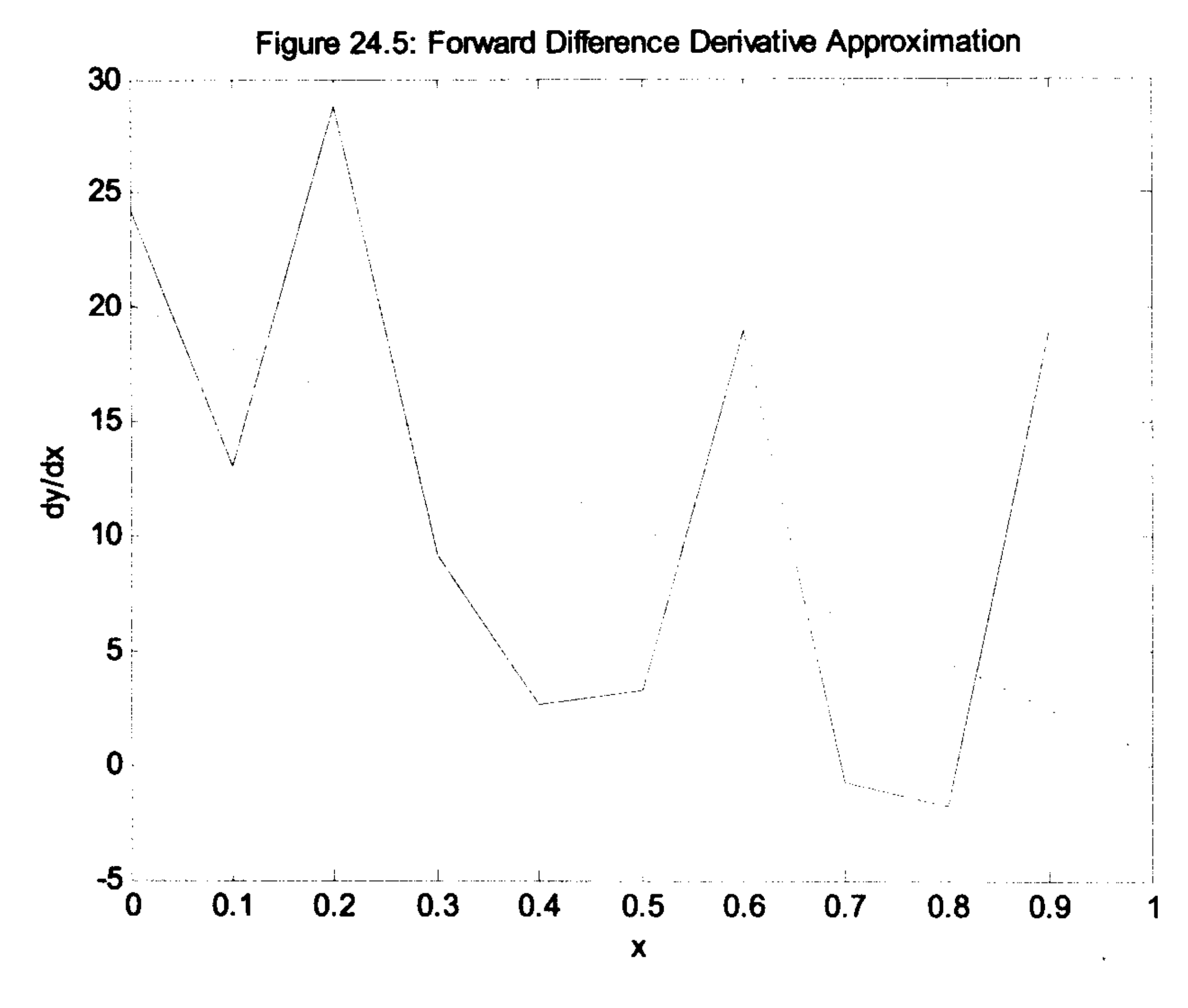


图 24.5　前向差分导数估计

由于 diff 函数是通过计算前向差分来估计导数值的，因此它所得到的结果数组的长度要比 x 或 y 的长度少 1。为了画出导数曲线，x 数组中的一个元素必须被删除，以使得结果数组与 x 数组长度相等。在实际操作时，有两种删除元素的方法：一种方法是删除 x 数组的第一个元素，这样，上边的求导过程就是一个后向微分估计过程，即用 x(n−1)和 x(n)点的差值来估计 x(n)点处的导数；另一种方法是删除 x 数组的最后一个元素，这样，上边的求导过程就是一个前向微分估计过程，即用 x(n+1)点和 x(n)点的差值来估计 x(n)点处的导数。应注意的是，删除 x 数组的一个元素仅仅是为了显示需要而采取的步骤，在估计导数时这些被删除的元素是必须被用到的。由图 24.5 可知，用 diff 求得的导数与用多项式近似求得的导数相比，其精确度差得很远，效果也很差。不过，当原始数据不存在不确定性时，用 diff 求得的导数结果还是可以接受的，尤其是我们在观察一个确定性函数导数时，使用 diff 显得既简便又快捷。下面给出了一个具体的例子：

```
>> x = linspace(0,2*pi);
>> y = sin(x);
>> dy = diff(y)/(x(2)-x(1));
```

```
>> xd = x(2:end);
>> plot(x,y,xd,dy)
>> axis tight
>> xlabel('x'),ylabel('sin(x) and cos(x)')
>> title('Figure 24.6: Backward Difference Derivative Approximation')
```

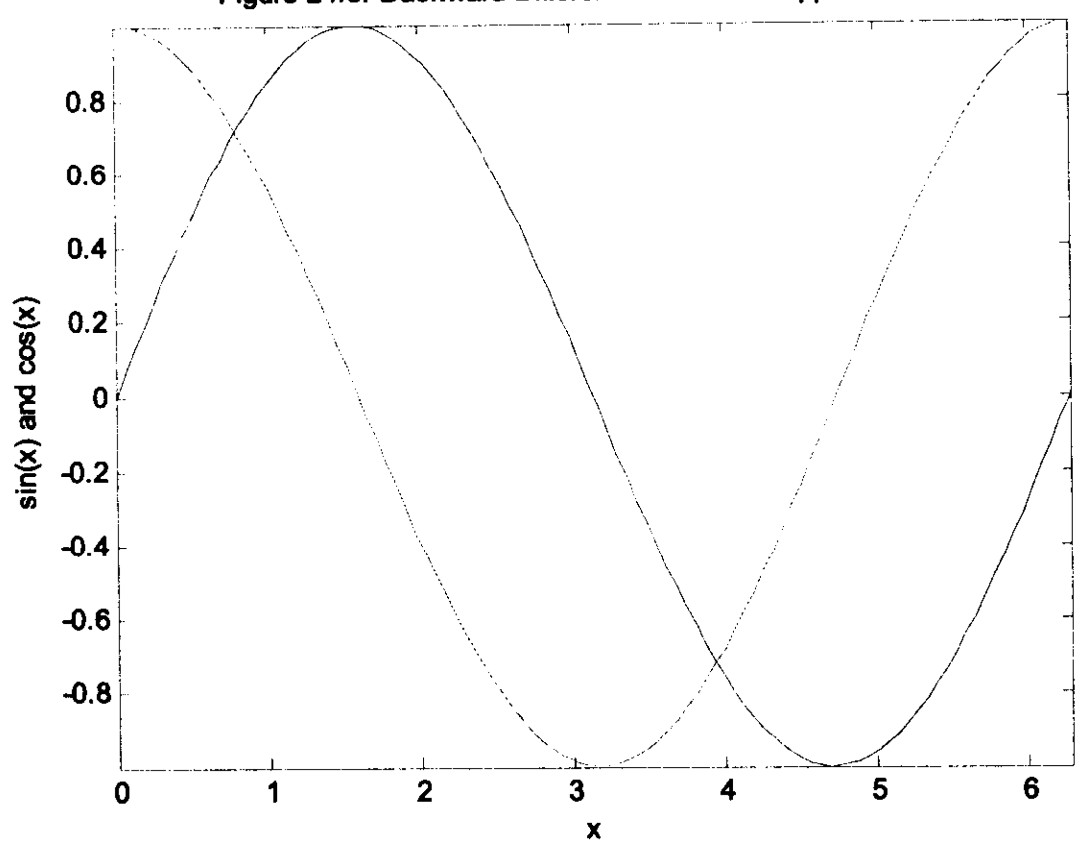


图 24.6 后向差分导数估计

上例中，x 是均匀间隔的，因此求导公式中用 x(2)-x(1)替代了 diff(x)，当 x 不是均匀间隔时，在求导公式中就不能用 x(2)-x(1)来替代 diff(x)了。另外，由于 x 的第一个元素被删除（xd = x(2:end);），因此它是一个后向差分导数估计过程。由图 24.6 可知，利用 diff 求得的导数还是非常准确的，我们可以利用下面的语句看一下它与实际导数函数的最大误差：

```
>> max(abs(cos(xd)-dy))
ans =
    0.0317
```

除了前向差分和后向差分外，用户还可以使用中心差分来近似估计导数。下面给出了一阶中心差分求导的公式：

$$\frac{\mathrm{d}y(x_n)}{\mathrm{d}x} \approx \frac{f(x_{n+1}) - f(x_{n-1})}{x_{n+1} - x_{n-1}}$$

因此，在使用一阶中心差分求导时，x_n 处的斜率就是其前后相邻数据点差值的函数。下面的代码给出了利用一阶中心差分估计上例 sin 函数的导数：

```
>> dy = (y(3:end)-y(1:end-2))/(x(3)-x(1));
>> xd = x(2:end-1);
>> max(abs(cos(xd)-dy))
ans =
    0.00067086
```

上例中，第一个和最后一个数据点没有对应的一阶中心差分导数估计，这是因为这两个数据点不存在前后两个相邻的数据点。不过，在所有的中间点，一阶中心差分估计导数的精确度要比前向或者后向差分导数估计值高大约两个数量级。

除了一维导数，Matlab 还提供了函数 gradient 估计二维数据的导数。该函数使用中心差分来估计二维数据列表中每个数据点在每一个方向上的斜率。由于第一个数据点和最后一个数据点不存在两个相邻的数据点，因此该函数在第一个数据点处使用了前向差分，在最后一个数据点处使用了后向差分，以保证求出的结果和原始数据点的长度一样。函数 gradient 的一个重要用途是图形数据可视化，如下面的代码所示：

```
>> [x,y,z] = peaks(20);     % simple 2-D function
>> dx = x(1,2) - x(1,1);    % spacing in x direction
>> dy = y(2,1) - y(1,1);    % spacing in y direction

>> [dzdx,dzdy] = gradient(z,dx,dy);

>> contour(x,y,z)
>> hold on
>> quiver(x,y,dzdx,dzdy)
>> hold off
>> title('Figure 24.7: Gradient Arrow Plot')
```


图 24.7　箭头梯度图

在上边的例子中，首先用 peaks 函数生成一个简单的二维数组，然后利用 gradient 函数计算这些数据的 dz/dx 和 dz/dy，并将其提供给函数 quiver，绘制出箭头梯度图。箭头梯度图以数组中每一点的表面法线为方向，绘制代表梯度的箭头图形，其中箭头的长度与梯度的大小成正比。

除了梯度之外，有时候用户也关心一个表面的曲率（即斜率的变化）。在 Matlab 中，每个点的曲率是用函数 del2 计算的，该函数求如下式所示的拉普拉斯变换的离散估计：

$$\nabla^2 z(x,y) = \frac{\mathrm{d}^2 z}{\mathrm{d}x^2} + \frac{\mathrm{d}^2 z}{\mathrm{d}y^2}$$

表面曲率一个最简单的计算方法是获取每个表面元素的值，然后用它减去与它相邻的 4 个元素的平均值。如果一个表面在给定点处的任何方向上都是平坦的，那么表面元素的值就不会发生变化。Matlab5 以后的版本在计算曲面内部点上的曲率时都使用中心二次差分来获得更精确的结果。下面给出了一个求表面曲率的例子：

```
>> [x,y,z] = peaks;          % default output of peaks
>> dx = x(1,2) - x(1,1);     % spacing in x direction
>> dy = y(2,1) - y(1,1);     % spacing in y direction
>> L = del2(z,dx,dy);
>> surf(x,y,z,abs(L))
>> shading interp
>> title ('Figure 24.8: Discrete Laplacian Color')
```

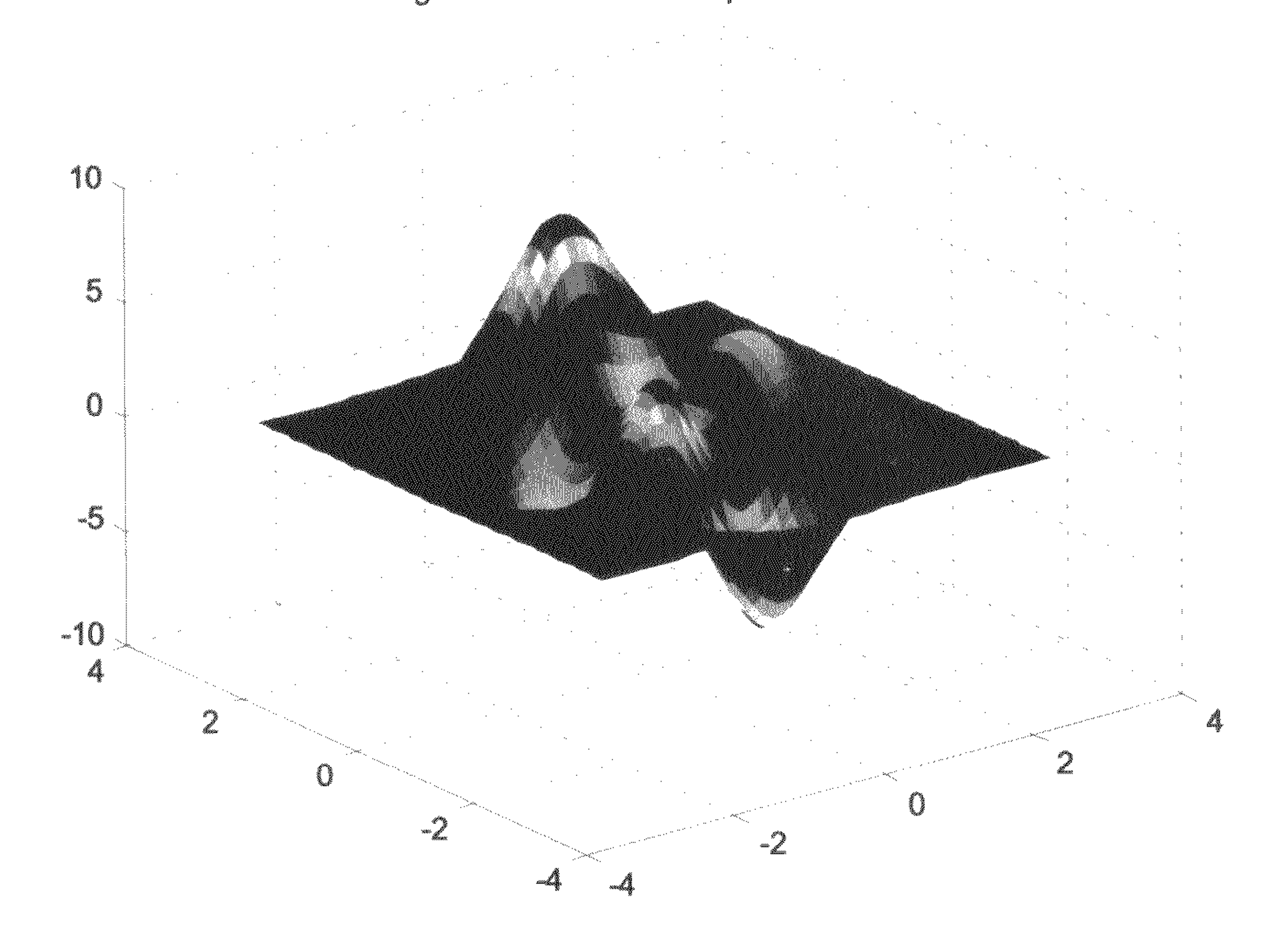


图 24.8 离散拉普拉斯变换

在图 24.8 中，根据表面曲率的不同，曲面将呈现出不同的颜色。

Chapter 25

微 分 方 程

微分方程，在数字信号处理领域又叫差分方程，是微积分数学和数字信号处理的重要基础。1995 年，Matlab 引入了一组 M 文件，称为 Matlab ODE 组，主要用来求解常微分方程（ODE）。在 Matlab 5 中，Matlab ODE 组已经成为了 Matlab 的一个标准组件。在 Matlab 6 中，ODE 组中又新增了两个初值问题（IVP）的解法程序。另外，Matlab 6 中还增加了几个函数用来求解边界值问题（BVP）和偏微分方程（PDE）。Matlab 6 以后，ODE 组又增加了求解延迟微分方程（DDE）和隐微分方程（IDE）的功能。

总的来说，目前 Matlab 能够解决有关微分方程的各种问题，但本章的目的不是对这些问题进行一一阐述，而只介绍了在实际应用中经常遇到的 ODE 初值问题（IVP）。

25.1 IVP 格式

Matlab 中的初值问题解法程序根据给定的初值条件，计算一系列相互关联的一阶微分方程的历史解。从数学角度上讲，该问题可以用如下形式描述：

$$\dot{y} = f(t,y) \qquad y(t_0) = y_0$$

上式是一种向量表达方式，它可以展开为如下的微分方程组：

$$\begin{aligned} \dot{y}_1 &= f_1(t, y_1, y_2, \dots, y_n) & y_1(t_0) &= y_{10} \\ \dot{y}_2 &= f_2(t, y_1, y_2, \dots, y_n) & y_2(t_0) &= y_{20} \\ &\vdots & &\vdots \\ \dot{y}_n &= f_n(t, y_1, y_2, \dots, y_n) & y_n(t_0) &= y_{n0} \end{aligned}$$

其中，$\dot{y}_i = \mathrm{d}y_i / \mathrm{d}t$，$n$ 是一阶微分方程的个数，y_{i0} 是第 i 个方程的初始条件。如果一个初值问题不是用一阶微分方程组的形式表示的，我们也必须利用初值将其转化为一阶。例如，考虑下面的经典 Van der Pol 方程：

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0$$

其中，μ 是一个大于 0 的参数。如果我们选择 $y_1=x$，$y_2=\mathrm{d}x/\mathrm{d}t$，则上述方程可转化为：

$$\dot{y}_1 = y_2$$
$$\dot{y}_2 = \mu(1 - y_1^2)y_2 - y_1$$

本章中，我们将以上面的经典 Van der Pol 方程为例，阐述 IVP 问题的求解方法。

25.2 ODE 组的解法程序

Matlab ODE 组提供了 8 个初值问题的解法程序，每个程序都适用于不同类型的初值问题。不过，每个解法程序的调用语法都是一样的，因此，对于一个给定的问题，我们可以很灵活地选择不同的解法程序进行验证。下表给出了这 8 个解法程序的具体描述：

解法程序	描述
ode23	显式的单步 Runge-Kutta 低阶（2 阶到 3 阶）解法程序。适用于具有一定难度的问题，对精确度要求不高的问题，或者 $f(t,y)$不平滑（比如，非连续）的问题
ode23s	隐含的单步修正 Rosenbrock 二阶解法程序。适用于对精确度要求不高的高难度问题，或者 $f(t,y)$不连续的高难度问题。高难度问题指隐含的时间常数的变化范围是其最小值的若干个数量级的问题
ode23t	使用自由插值的隐含单步梯形规则。适用于稍有难度的问题。可以用来求解微分一代数方程（DAE）
ode23tb	附带了二阶后向微分算法的隐含梯形规则。与 ode23s 类似。对于误差容限要求不高的场合，要比 ode15s 的效率高
ode45	显式的单步 Runge-Kutta 中阶（4 阶或者 5 阶）解法程序。适用于对精确度有一定要求的非难度问题。当用户求解一个新问题时，通常首选这一解法程序
ode113	不同阶次（从 1 阶到 13 阶）的多步 Adams-Bashforth-Moulton PECE 解法程序。适用于对精确度有一定要求或较高要求，且 $f(t,y)$计算时开销很大的非难度问题。不适用于 $f(t,y)$不平滑的问题
ode15s	不同阶次（1 阶到 5 阶）的隐含多步数值微分解法程序。适用于对精确度要求适中的高难度问题。通常当 ode45 失效或效率太低时，用户就选择用该解法程序求解。
ode15i	一个用于求解完全隐微分方程的不同阶次（1 阶到 5 阶）的解法程序。

上面表格中所用到的一些术语，如显式、隐含、难度等，都需要参考大量的基础知识才能理解。如果用户能够很好地理解这些术语，就比较容易理解每个解法程序的基本属性；即使用户不能完全理解这些术语，只要用户按照表格中给出的适用范围能够应用这些解法程序就可以了。其中有一个重要的小技巧，就是当用户首次求解一个指定的问题是，最好先选用 ode45，若不行，再选用 ode15s，实在不行，再选用别的解法程序。

值得注意的是，Matlab ODE 组是以一组可以浏览的 M 文件的形式提供的。并且，这些解法程序也被包含在了 SIMULINK 模块中，用户可以直接用其来对动态系统进行仿真。

25.3 基本用法

我们在求解一个微分方程组之前，必须首先为该方程组创建一个 M 函数文件。该文件将接受一个时间向量 t 和解向量 y，返回导数值。对于 25.1 节的经典 Van der Pol 方程，我们可以用下面的 M 文件函数来表示：

```
function ydot=vdpol(t,y)
%VDPOL van der Pol equation.
% Ydot=VDPOL(t,Y)
% Ydot(1) = Y(2)
% Ydot(2) = mu*(1-Y(1)^2)*Y(2)-Y(1)
% mu = 2

mu = 2;
ydot = [y(2); mu*(1-y(1)^2)*y(2)-y(1)];
```

在上面的函数定义中，虽然提供了两个输入参数 t 和 y，但函数并没有用到 t。另外，注意输出 ydot 必须是一个列向量。

下面给出了经典 Van der Pol 方程的求解代码：

```
>> tspan = [0 20];      % time span to integrate over

>> yo = [2; 0];         % initial conditions (must be a column)
>> [t,y] = ode45(@vdpol,tspan,yo);
>> size(t)              % number of time points
ans =
   333     1
>> size(y)              % (i)th column is y(i) at t(i)
ans =
   333     2
>> plot(t,y(:,1),t,y(:,2),'--')
>> xlabel('time')
>> title('Figure 25.1: van der Pol Solution')
```

上例在调用 ode45 函数时使用函数句柄是推荐使用的方法，因为 ode45 函数在得到一合理的解的过程中需要多次运行微分方程函数（vdpol），而对于需要反复运行的函数，传递句柄要比直接传递函数名节省很多执行时间。

在默认情况下，如果一个解法程序在调用时没有输出参数，比如 ode45(@vdpol,tspan,yo)，则该程序就不会生成输出变量，但会画出与图 25.1 相似的时间图形。如果一个解法程序在调用时有一个输出参数，该程序就会将一个包含日后验证算法结果（如使用函数 deval 验证时）所需要的所有信息的结构体返回到惟一的输出参数中。下面给出了 ode45 的一个输出参数的例子：

```
>> sol = ode45(@vdpol, tspan, yo)
sol =
         x: [1×84 double]
         y: [2×84 double]
```

```
solver: 'ode45'
 idata: [1×1 struct]
```



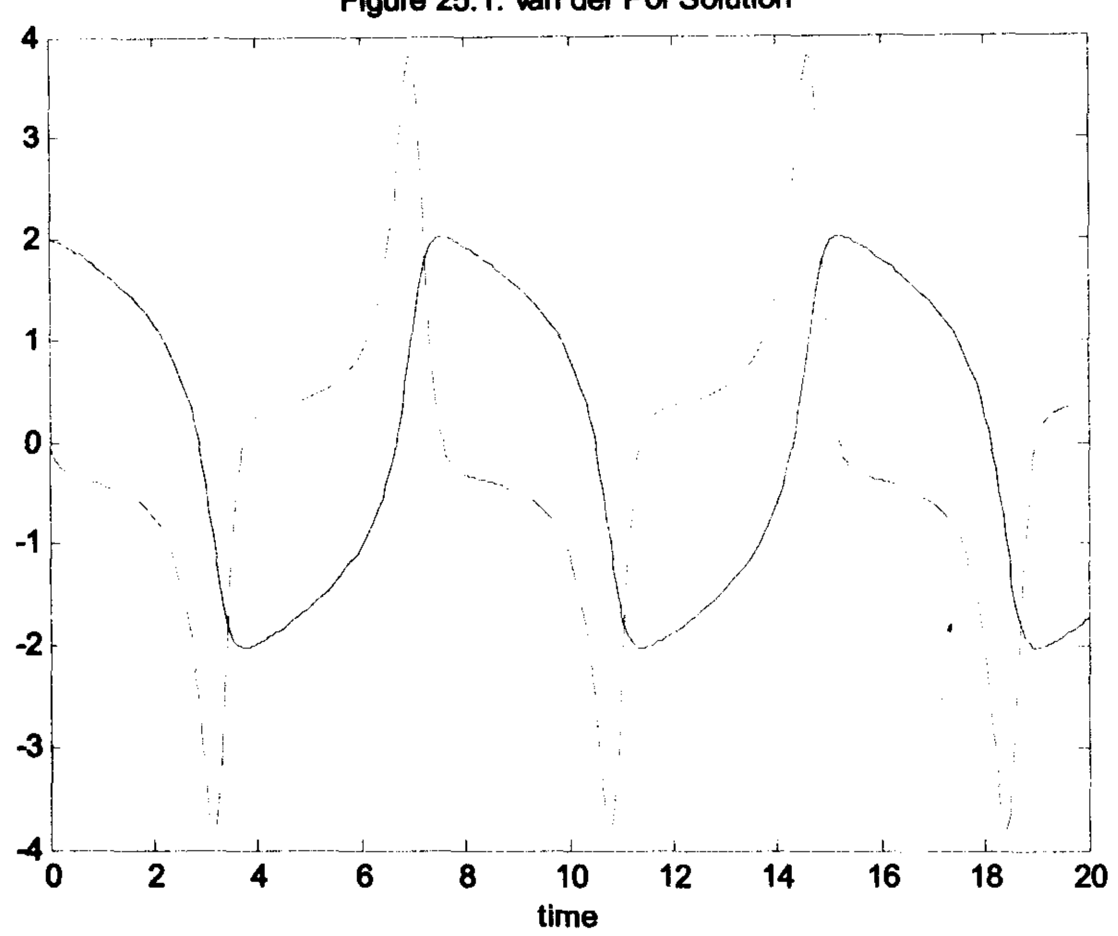


图 25.1 van der Pol 方程的解

deval 函数的帮助文档里有关于 sol 的详细信息。另外，根据 sol 结构体，用户还可以使用函数 odextend 对求得的结果进行推广。

用户除了可以在 tspan 参数中声明起始时间和结束时间之外，还可以指定要求解的时间点。例如，下面的代码在 100 个时间点上求解微分方程 vdpol：

```
>> yo = [2; 0];
>> tspan = linspace(0,20,100);
>> [t,y] = ode45(@vdpol,tspan,yo);
>> size(t)
ans =
    100     1
>> size(y)
ans =
    100     2
```

在采用上述方式调用 ode45 的时候，解法程序仍旧使用自动步长控制来保证精确度。注意：任何解法程序都不会采用固定步长进行积分。解法程序通常会对自己求出的解进行插值处理，在不降低解的精确度的前提下尽可能获得更多的求解信息。

有时候微分方程组中需要包含若干个便于用户控制的参数集。这些参数一般存在于解法程序和 ODE 文件的输入参数中，这样就不需要在每次调用之前打开文件来修改其中的参数。例如，我们可以将 vdpol 函数中的 μ 设为用于可调整的输入参数，重新修改 vdpol 函数如下：

```
function ydot=vdpol(t,y,mu)
%VDPOL van der Pol equation.
% Ydot=VDPOL(t,Y,mu)
% Ydot(1) = Y(2)
% Ydot(2) = mu*(1-Y(1)^2)*Y(2)-Y(1)

% mu = ?; now passed as an input argument

if nargin<3 % supply default if not given
    mu=2;
end
ydot = [y(2); mu*(1-y(1)^2)*y(2)-y(1)];
```

当调用上面的函数时，μ 就成了用户可控制的参数，如下例所示：

```
>> mu = 10 % set mu in Command window
mu =
    10
>> ode45(@vdpol,tspan,yo,[],mu)
>> title('Figure 25.2: van der Pol Solution (\mu=10) ')
```

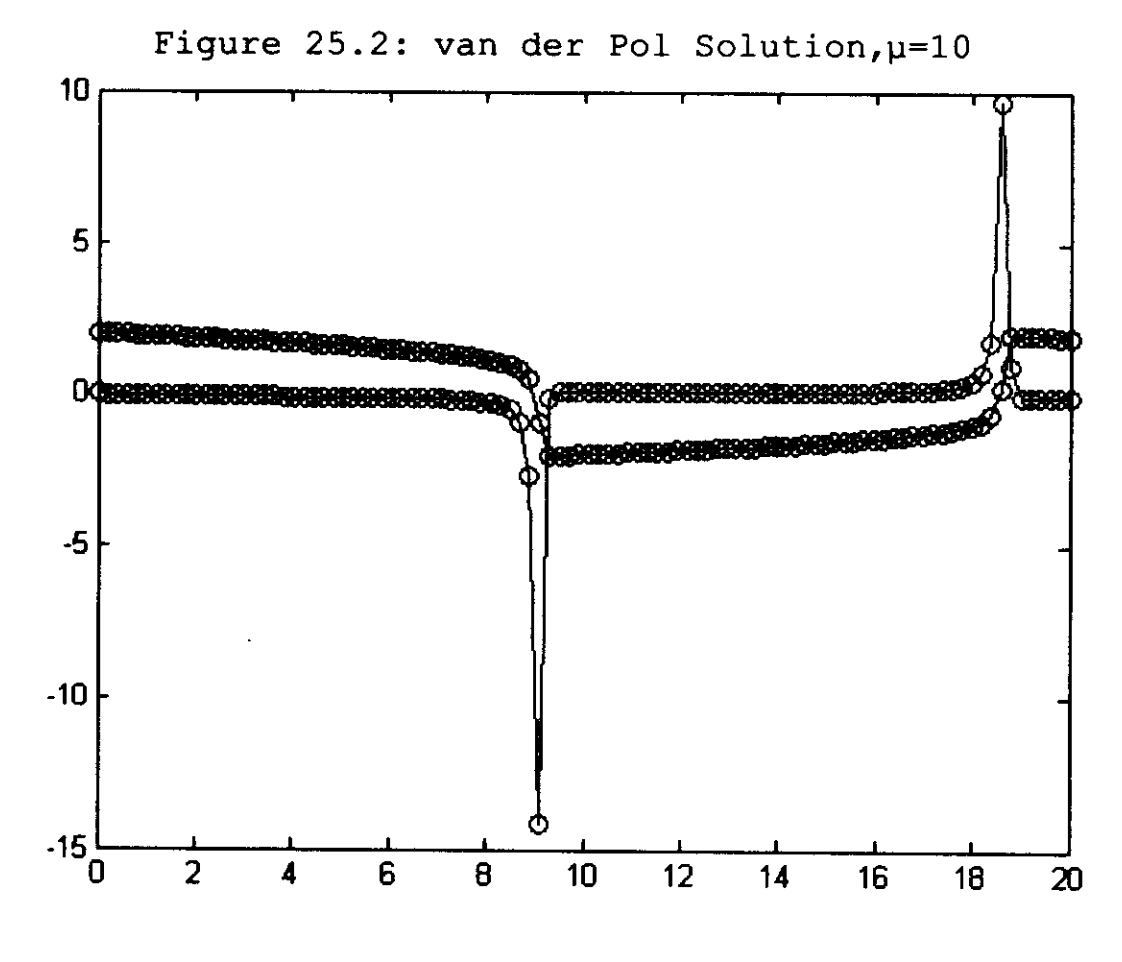


图 25.2 van der Pol 方程的解（μ=10）

上例在 μ=10 的条件下重新求解了 van der Pol 方程组。在 ode45 的调用方式中多出了两个输入参数：第一个参数是一个空数组，它告诉 ode45 函数使用默认的解法程序选项，第二个参数用于设置 μ 的值。另外，本例中 ode45 采用了无输出参数的调用方式，因此会自动绘制出解的图形。

虽然在上边的例子中，我们只添加了一个输入参数，但实际上用户可以在 vdpol 函数中添加任意数量的输入参数，只要在 ode45 调用时将这些参数考虑在内即可。另外，Matlab 对添加的输入参数的类型没有限制。

25.4 设置选项

Matlab ODE 组也提供了一个选项结构允许用户在进行计算或显示时采用个性化的方式，比如指定误差容限等。我们在前面的函数调用中都使用了解法程序的默认误差容限和选项。当需要进行个性化操作时，就需要将一个选项结构作为解法程序的第四个输入参数传递给它。这个选项结构通常由函数 odeset 和 odeget 来管理。odeset 的工作方式类似于句柄图形的 set 函数，其参数也是以名称/属性值对的形式声明的。例如，options=odeset ('Name1',value1, 'Name2', value2,...)。odeset 的帮助文档提供了所有的参数名称及其属性值的列表：

```
>> help odeset
ODESET Create/alter ODE OPTIONS structure.
   OPTIONS=ODESET('NAME1',VALUE1,'NAME2',VALUE2,...) creates an integrator
    options structure OPTIONS in which the named properties have the
    specified values. Any unspecified properties have default values. It is
    sufficient to type only the leading characters that uniquely identify
    the property. Case is ignored for property names.
    OPTIONS = ODESET(OLDOPTS,'NAME1',VALUE1,...) alters an existing options
    structure OLDOPTS.
    OPTIONS = ODESET(OLDOPTS,NEWOPTS) combines an existing options structure
    OLDOPTS with a new options structure NEWOPTS. Any new properties
    overwrite corresponding old properties.
    ODESET with no input arguments displays all property names and their
    possible values.
ODESET PROPERTIES
RelTol - Relative error tolerance  [ positive scalar {1e-3} ]
    This scalar applies to all components of the solution vector, and
    defaults to 1e-3 (0.1% accuracy) in all solvers.  The estimated error in
    each integration step satisfies e(i) <= max(RelTol*abs(y(i)),AbsTol(i)).
AbsTol - Absolute error tolerance  [ positive scalar or vector {1e-6} ]
    A scalar tolerance applies to all components of the solution vector.
    Elements of a vector of tolerances apply to corresponding components of
    the solution vector. AbsTol defaults to 1e-6 in all solvers. See RelTol.
NormControl -  Control error relative to norm of solution  [ on | {off} ]
    Set this property 'on' to request that the solvers control the error in
    each integration step with norm(e) <= max(RelTol*norm(y),AbsTol). By
    default the solvers use a more stringent component-wise error control.
Refine - Output refinement factor  [ positive integer ]
    This property increases the number of output points by the specified
    factor producing smoother output. Refine defaults to 1 in all solvers
    except ODE45, where it is 4. Refine does not apply if length(TSPAN)>2
    or the ODE solver returns the solution as a structure.
OutputFcn - Installable output function  [ function ]
    This output function is called by the solver after each time step. When
    a solver is called with no output arguments, OutputFcn defaults to the
    function odeplot. Otherwise, OutputFcn defaults to [].
OutputSel - Output selection indices  [ vector of integers ]
```

```
      This vector of indices specifies which components of the solution vector
      are passed to the OutputFcn. OutputSel defaults to all components.
   Stats - Display computational cost statistics  [ on | {off} ]
   Jacobian - Jacobian function [ function | constant matrix ]
      Set this property to a function FJac(if FJac(t,y) returns dF/dy) or to
      the constant value of dF/dy.
   JPattern - Jacobian sparsity pattern [ sparse matrix ]
      Set this property to a sparse matrix S with S(i,j) = 1 if component i
      of F(t,y) depends on component j of y, and 0 otherwise.
   Vectorized - Vectorized ODE function  [ on | {off} ]
      Set this property 'on' if the ODE function F is coded so that
      F(t,[y1 y2 ...]) returns [F(t,y1) F(t,y2) ...].
   Events - Locate events  [ function ]
      To detect events, set this property to the event function.
   Mass - Mass matrix [ constant matrix | function ]
      For problems M*y' = f(t,y) set this property to the value of the constant
      mass matrix. For problems with time- or state-dependent mass matrices,
      set this property to a function that evaluates the mass matrix.
   MStateDependence - Dependence of the mass matrix on y [ none | {weak} | strong ]
      Set this property to 'none' for problems M(t)*y' = F(t,y). Both 'weak'
      and 'strong' indicate M(t,y), but 'weak' will result in implicit solvers
      using approximations when solving algebraic equations.
   MassSingular - Mass matrix is singular  [ yes | no | {maybe} ]
      Set this property to 'no' if the mass matrix is not singular.
   MvPattern - dMv/dy sparsity pattern [ sparse matrix ]
      Set this property to a sparse matrix S with S(i,j) = 1 if for any k, the
      (i,k) component of M(t,y) depends on component j of y, and 0 otherwise.
   InitialSlope - Consistent initial slope yp0 [ vector ]
      yp0 satisfies M(t0,y0)*yp0 = F(t0,y0).
   InitialStep - Suggested initial step size  [ positive scalar ]
      The solver will try this first.  By default the solvers determine an
      initial step size automatically.
   MaxStep - Upper bound on step size  [ positive scalar ]
      MaxStep defaults to one-tenth of the tspan interval in all solvers.
   BDF - Use Backward Differentiation Formulas in ODE15S  [ on | {off} ]
      This property specifies whether the Backward Differentiation Formulas
      (Gear's methods) are to be used in ODE15S instead of the default
      Numerical Differentiation Formulas.
   MaxOrder - Maximum order of ODE15S [ 1 | 2 | 3 | 4 | {5} ]
```

如果用户在命令窗口中输入一个单独的 odeset 命令，就会出现一个如下所示的简化的选项参数列表：

```
>> odeset
            AbsTol: [ positive scalar or vector {1e-6}]
            RelTol: [ positive scalar {1e-3} ]
       NormControl: [ on | {off} ]
         OutputFcn : [ function ]
         OutputSel : [ vector of integers ]
            Refine: [ positive integer ]
             Stats: [ on | {off} ]
```

```
        InitialStep: [ positive scalar ]
            MaxStep : [ positive scalar ]
                BDF : [ on | {off} ]
           MaxOrder : [ 1 | 2 | 3 | 4 | {5} ]
           Jacobian : [ matrix | function ]
           JPattern : [ sparse matrix ]
         Vectorized: [ on | {off} ]
               Mass : [ matrix | function ]
  MStateDependence: [ none | {weak} | strong ]
          MvPattern : [ sparse matrix ]
       MassSingular: [ yes | no | {maybe} ]
       InitialSlope: [ vector ]
             Events: [ function ]
```

在上述的参数列表中，中括号给出的是各参数可以取的值，大括号中给出的是参数的默认值。下面给出的是设置选项结构中的 AbsTol 和 RelTol 进行差分方程求解的一个例子：

```
>> tspan = [0 20]; % set time Span to solve
>> yo = [2; 0];     % intial conditions
>> mu = 10;         % parameter mu
>> options = odeset('AbsTol',1e-12,'RelTol',1e-6);

>> [t,y] = ode45(@vdpol,tspan,yo,[],mu);  % default tolerances
>> length(t)
ans =
    593
>> [t,y] = ode45(@vdpol,tspan,yo,options,mu);  % tight tolerances
>> length(t)
ans =
    1689

>> [t,y] = ode15s(@vdpol,tspan,yo,[],mu);  %  new solver, default tols
>> length(t)
ans =
    232
>> [t,y]= ode15s(@vdpol,tspan,yo,options,mu); %  new solver, tight tols
>> length(t)
ans =
    651
```

上述结果说明，不同的误差容限将导致不同的执行时间。在默认容限下，即 AbsTol＝1e-6，RelTol=1e-3，使用 ode45 需要 593 次运算才能完成，如果将容限降到 AbsTol＝1e-12 和 RelTol=1e-6，就需要 1689 次运算。另外，在相同的容限下，使用 ode15s 要比 ode45 节省将近 3 倍的时间。

选项结构中的另一个重要参数是 Jacobian 参数。差分方程的解法程序 ode15s、ode23s、ode23t 和 ode23tb 都提供了对 Jacobian 矩阵的支持，不过，数值近似仍是其默认的设置。Jacobian 矩阵是一个偏导数矩阵，其格式如下：

$$\begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots & \frac{\partial f_1}{\partial y_n} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \cdots & \frac{\partial f_2}{\partial y_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial y_1} & \frac{\partial f_n}{\partial y_2} & \cdots & \frac{\partial f_n}{\partial y_n} \end{bmatrix}$$

高难度解法程序通常利用该矩阵或对其的数值近似来计算一个非线性方程组在各个时间步长上的解。下面给出了 Jacobian 矩阵的 M 函数文件：

```
function jac=vdpoljac(t,y,mu)
%VDPOLJAC van der Pol equation Jacobian.

% mu = ?; passed as an input argument
if nargin<3 % supply default if not given
    mu=2;
end
jac = [      0                      1
        (-2*mu*y(1)*y(2)-1)  (mu*(1-y(1)^2))];
```

如果要利用 Jacobian 矩阵进行微分方程求解，需要利用 odeset 函数调整选项结构的设置，然后再调用相应的 ODE 组函数。下面给出了一个应用实例：

```
>> options = odeset(options,'Jacobian',@vdpoljac);
>> [t,y] = ode15s(@vdpol,tspan,yo,options,mu);
>> length(t)
ans =
    670
```

上例中，首先利用 odeset 函数将前面的 Jacobian 矩阵文件的函数句柄添加到一个选项结构设置中，然后调用了 ode15s 执行微分方程的求解。从运行情况看，虽然此次求解共执行了 670 次运算（比前面的 651 次多出 19 次），但由于是采用 Jacobian 矩阵的方式求解的，因此，实际的运行速度却得到了提升。

在用户无法提供一个解析的 Jacobian 矩阵时，提供一个向量化的 ODE 文件是不无裨益的。将 ODE 文件向量化就意味着将 y 由一维向量变为二维数组，也就是说，要用 y(i,:)代替 y(i)，并且将其中的向量操作符换为数组操作符。将 ODE 文件向量化可以使 Matlab ODE 组函数按照数值类型的 Jacobian 矩阵的方式求解，从而提高运行速度。

下面的文件给出了向量化后的 van der Pol 方程：

```
function ydot=vdpol(t,y,mu)
%VDPOL van der Pol equation.
% Ydot=VDPOL(t,Y)
% Ydot(1) = Y(2)
% Ydot(2) = mu*(1-Y(1)^2)*Y(2)-Y(1)

% mu = ?; now passed as an input argument
```

```
if nargin<3 % supply default if not given
    mu=2;
end
ydot = [y(2,:); mu*(1-y(1,:)^2)*y(2,:)-y(1,:)];
```

选项结构中的 Refine 参数决定了需要产生多少个输出数据。它不会影响解法程序选择的步长大小（即精确度），仅仅规定在每一次积分步骤中，需要在解中插入多少个中间点。下面给出了一个具体例子：

```
>> options = odeset('Refine',1);
>> [t,y] = ode45(@vdpol,tspan,yo,options,mu);
>> length(t) % # of time points
ans =
    149
>> options = odeset('Refine',4);
>> [t,y] = ode45(@vdpol,tspan,yo,options,mu);
>> length(t) % # of time points
ans =
    593
```

由程序执行结果可知，随着 refine 参数的增大，ode45 函数的执行时间也在增加，这是因为 ode45 需要在插入的中间点上进行运算。

选项结构中的 Events 参数允许用户对 ODE 求解过程中发生的一个或者多个事件进行标记。这里的事件指程序执行过程中遇到的一个状态，例如，某个解达到最大值、最小值或者穿过 0 点都属于简单的事件。用户可以通过选项结构的设置，使程序在某一事件发生时中止执行。要利用这个特性，用户需要提供一个事件函数，并利用 odeset 函数将其句柄传递给选项结构。在每一个计算步骤中，解法程序都检查这些事件，并及时标记事件发生的位置。下面给出了一个简单的事件函数，该函数检查 y 的绝对值是否为零，但不对函数执行产生影响：

```
function [value,isterminal,direction]=vdpolevents(t,y,mu)
%VDPOLEVENTS van der Pol equation events.
value(1)=abs(y(2))-2;    % find where |y(2)|=2
isterminal(1)=0;         % don't stop integration
direction(1)=0;          % don't care about crossing direction
```

事件函数 vdpolevents 接受与 ODE 函数相同的 3 个输入参数：t、y、mu，返回 3 个数值型向量。第一个向量 value 是事件的值；第二个向量 isterminal 是一个逻辑数组，它规定在事件发生时，解法程序是否应该终止程序的执行；第三个向量 direction 规定是否需要考虑事件发生的方向。vdpolevents 函数中，$abs(y_2(t))=2$ 被选为惟一的事件，并告诉解法程序在检测到事件发生时候不要停止执行，并且不考虑事件发生的方向。下面的代码是对该事件函数的一个简单应用：

```
>> mu =2;
>> options = odeset('Events',@vdpolevents);
>> [t,y,te,ye] = ode45(@vdpol,tspan,yo,options,mu);
```

```
>> plot(t,y,te,ye(:,2),'o')
>> title('Figure 25.3: van der Pol Solution (|y(2)|=2) ')
```

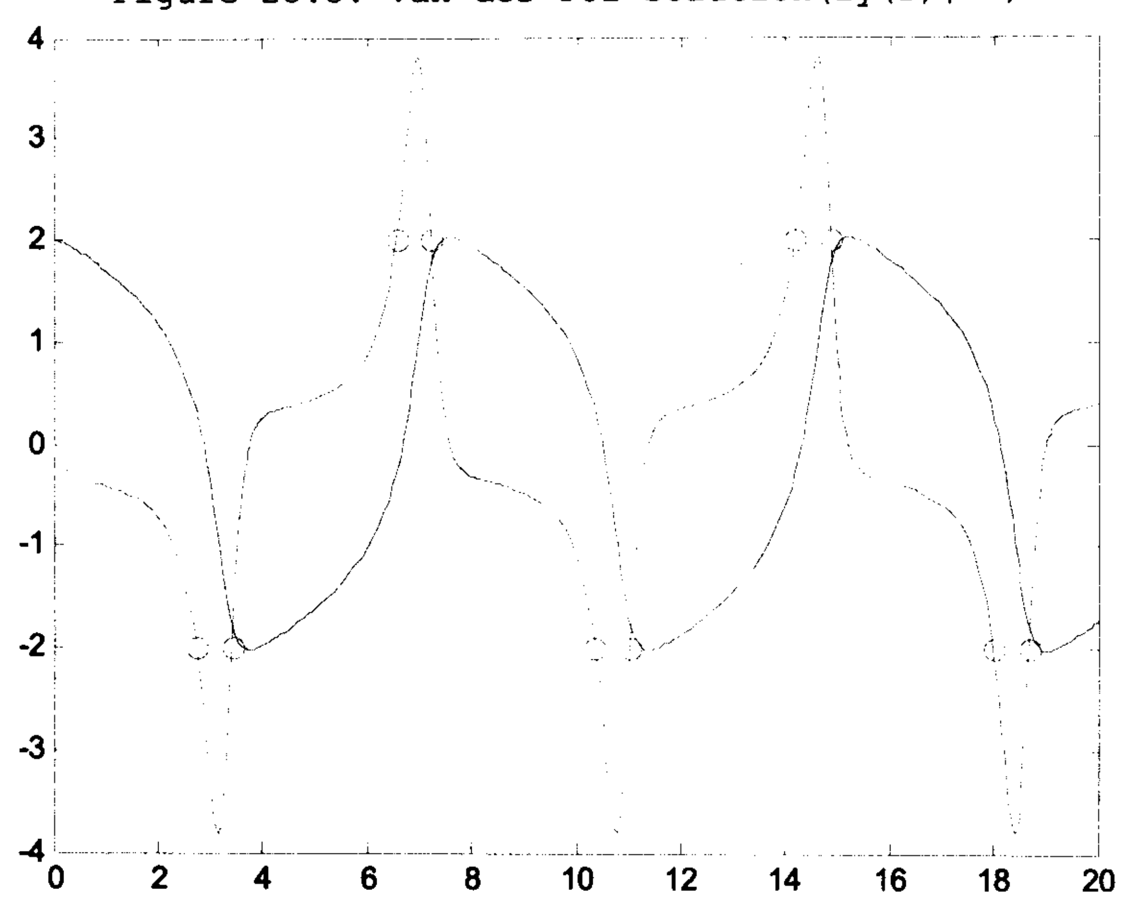


图 25.3 van der Pol 方程的解（|y(2)|=2）

图 25.3 中显示了事件发生的位置。

25.5 BVP、PDE 和 DDE

除了前几节介绍的初值微分方程（IVP）的 Matlab 解法程序外，Matlab 7 还提供了用来求解边界值问题（BVP）、偏微分方程（PDE）和延迟微分方程（DDE）的解法程序，本节我们不对它们进行详细阐述，仅以表格的形式对其进行总结。

下表给出了求解 BVP 的相关函数：

函数	描述
bvp4c	BVP 解法程序
bvpget	获得 BVP 选项结构
bvpinit	形成假设初始解，该值可以通过 bvp4c 进行修正
bvpset	设置 BVP 选项结构
deval	验证用 bvp4c 得到的解，或对这个解进行插值

下表给出了求解 PDE 的相关函数：

函数	描述
pdepe	求一维抛物线一椭圆 PDE 的 IVP 解
pdeval	验证用 pdepe 得到的解，或对这个解进行插值

下表给出了求解 DDE 的相关函数：

函数	描述
dde23	用于求解具有固定延迟的 DDE 初值问题
deval	验证用 dde23 得到的解，或对这个解进行插值
ddeget	从选项结构中获得 DDE 选项
ddeset	创建或改变 dde23 所使用的选项结构

有关以上表中各函数的详细信息，请用户参考相应的 Matlab 帮助文档。

Chapter 26

二 维 图 形

在前面的章节中，我们已经涉及到了 Matlab 的一些图形特性。在本章和后边的几章里，我们将对 Matlab 的图形特性进行更详细的描述。本书所阐述的很多图形特性和功能读者都能在图形窗口顶端的菜单栏中找到，另外，在图形和摄像机工具栏中也可以找到与图形特性和功能相关的按钮。图形和摄像机工具栏是在生成图形时默认显示的，如果找不到，读者也可以通过选择图形窗口中的 View 菜单栏中的 Figure/Camera 选项将其显示出来。

总之，Matlab 的图形窗口提供了一系列工具栏和菜单栏，使用户能够很方便地定制一幅图形。此外，用户也可以使用命令窗口函数使 Matlab 在绘图过程中实现定制。由于图形窗口的工具栏和菜单栏比较容易掌握，本书将集中讨论命令窗口函数，不过，这些函数所能实现的功能读者都可以利用相应的菜单栏和工具栏实现。

26.1 plot 函数

plot 函数是绘制二维图形最常用的函数。该函数将数组（或集合）中的数据绘制在相应的坐标平面上，并用直线将这些点连接起来，形成一幅连续的曲线图形。例如，下面的代码利用 plot 绘制出了 sin 函数的图形：

```
>> x = linspace(0,2*pi,30);
>> y = sin(x);
>> plot(x,y),title('Figure 26.1: Sine Wave')
```

上例在 $0 \leqslant x \leqslant 2\pi$ 的范围内生成了 30 个数据点作为图形的水平坐标轴，然后生成了另一个向量 y 包含与 x 相对应的正弦值。当 plot 函数被调用时，便打开一个称为图形窗口的视窗，将两个坐标轴进行适当缩放以满足数据绘制的要求，然后在指定的位置上绘制出数据点，最后将这些数据点用直线连接起来。plot 函数还能自动给坐标轴标注上坐标值以及标号。如果一个图形窗口已经存在，plot 函数就会自动将当前图形擦除然后绘制新的图形。

利用 plot 函数还可以在一幅图形上绘制多条曲线或直线，如下面的代码所示：

```
>> z = cos(x);
>> plot(x,y,x,z)
>> title('Figure 26.2:Sine and Cosine')
```

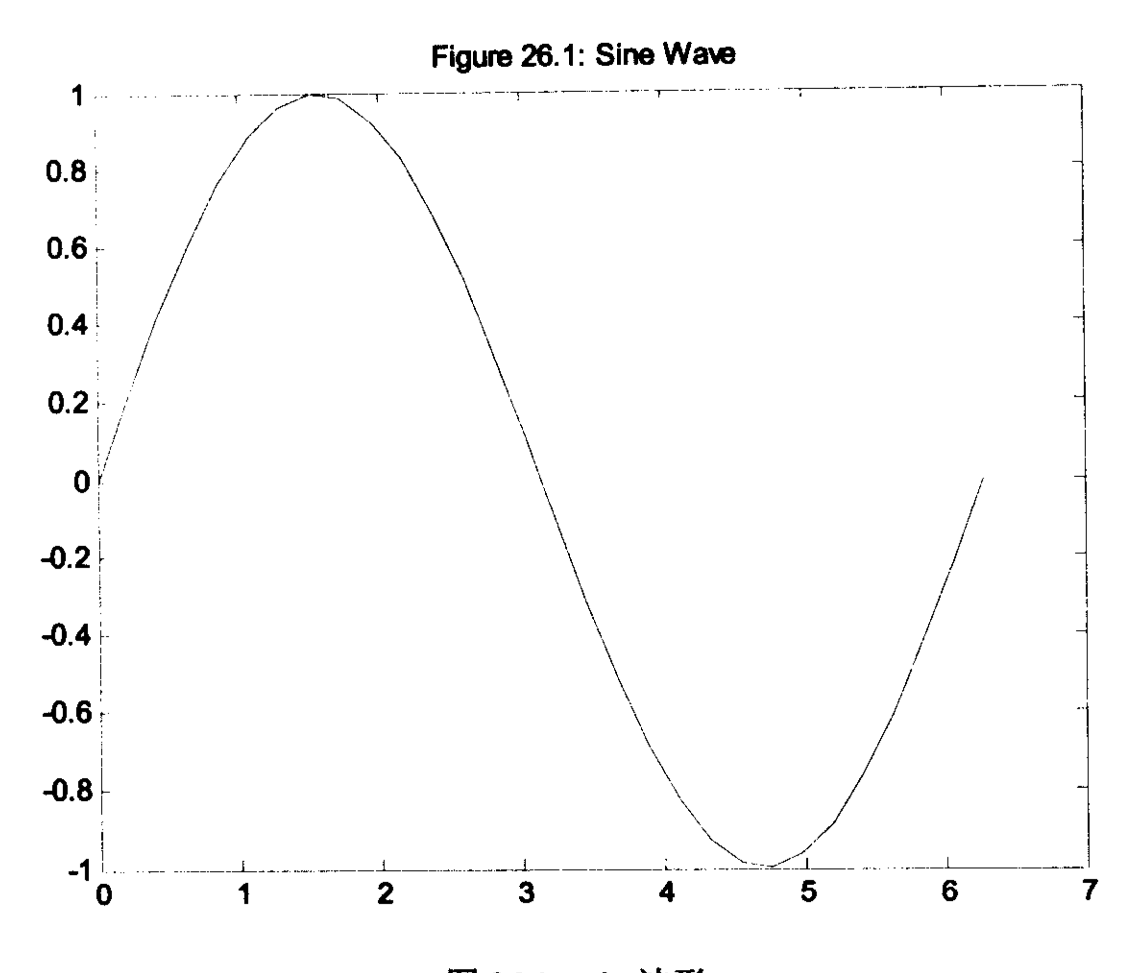


图 26.1　sin 波形

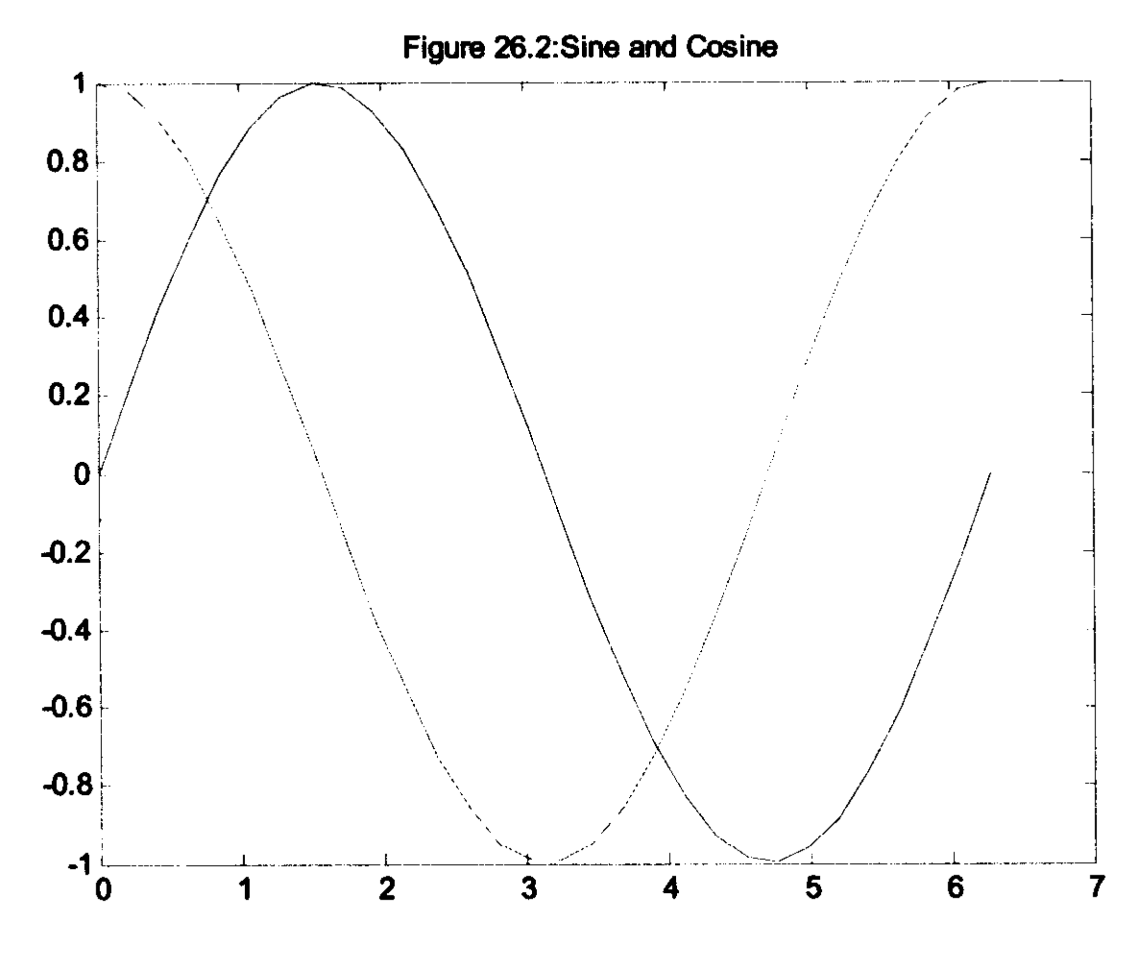


图 26.2　sin 和 cos 波形

只要在调用 plot 函数时给它提供第二个参数对（如，x-z），该函数就会绘制出另一条曲线。上例中，plot 函数接收到了两个参数对 x-y 和 x-z，就在同一幅图形上绘制出了 sin(*x*)和 cos(*x*)的波形。另外，由于打印的原因，有一点书中没有体现出来，就是 plot 函数会自动以不同的颜色来绘制每条曲线。实际上，对 plot 函数而言，能绘制的曲线个数是没有限制的，用户传递给它多少个输入参数对，它就会画出多少条曲线。

如果输入参数对一个是矩阵，一个是向量，则 plot 函数会将矩阵中的每一列与向量参

数组成一个输入对，分别绘制出相应的图形。例如，下面的代码先将 y 和 z 组合到一个矩阵中，然后调用 plot 函数进行绘图：

```
>> W = [y;z];        % create a matrix of the sine and cosine
>> plot(x,W)         % plot the columns of W vs. x
```

该 plot 命令将会得到与图 26.2 相同的图形。

如果用户改变了参数对的顺序，plot 函数就将绘制的图形旋转 90 度，如下面的代码所示：

```
>> plot(W,x)     % polt x vs. the columns of W
>> title('Figure 26.3: Change Argument Order')
```

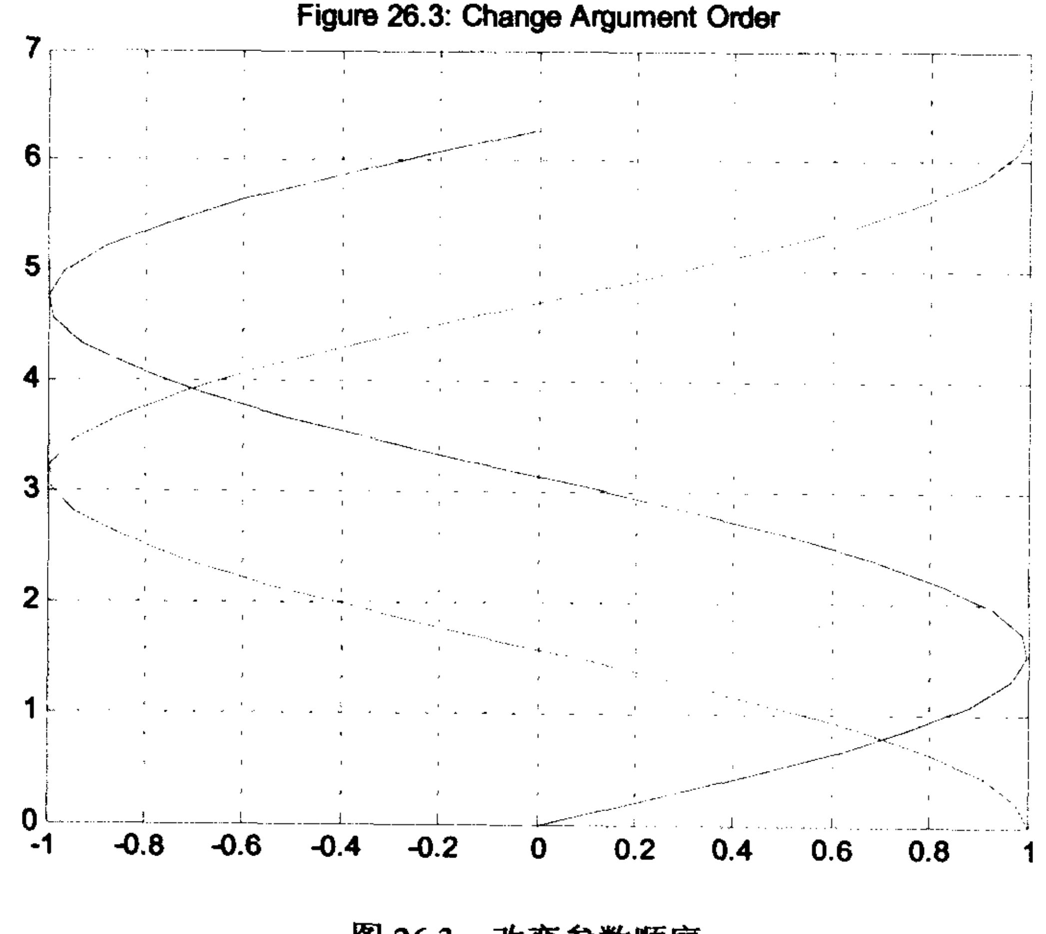


图 26.3 改变参数顺序

最后，读者也可以只用一个输入参数调用 plot 函数（例如 plot(Y)）。此时，plot 函数就会根据 Y 中包含的数据进行不同的操作：如果 Y 是一个复数向量，plot(Y)就相当于执行 plot(real(Y),imag(Y))（注意：只有在单输入参数情况下，plot 函数才会绘制复数的虚部图形，在其他情况下，输入向量的虚部是被忽略的）；如果 Y 是一个实值向量，那么 plot(Y)就相当于执行 plot(1:length(Y),Y)，也就是说，plot 函数将以 Y 的各元素的索引值为横坐标，以 Y 值为纵坐标绘制其图形；当 Y 是一个矩阵的时候，就将 Y 的每一列视为实值向量来处理。

26.2 线型、标记和颜色

在用户没有指定的情况下，Matlab 的绘图函数会默认地选择实线线型，并以一个默认的颜色顺序绘制每一个图形的颜色。不过，Matlab 的绘图函数是允许用户指定图形的线型和颜色的，另外，用户还可以指定用于区别数据点的标记。要实现这些定制化操作，用户只需将下表中的符号以字符串的形式传递给 Matlab 绘图函数就可以了：

符号	颜色	符号	标记	符号	线型
b	蓝色	.	点号	-	实线
g	绿色	o	圆圈	:	点线
r	红色	×	叉号	-.	点划线
c	青色	+	加号	--	虚线
m	洋红	*	星号		
y	黄色	s	方形		
k	黑色	d	菱形		
w	白色	∨	三角形（向下）		
		∧	三角形（向上）		
		<	三角形（向左）		
		>	三角形（向右）		
		p	五角星		
		h	六角星		

如果用户没有声明颜色并且在使用默认颜色机制，Matlab 就为每一条新增加的线从上边这个表格的前 7 种颜色的蓝色和圆圈开始从上到下依次选择颜色和标记类型。默认的线型是实线，除非用户显式地声明另一种线型。没有默认的标记。如果没有选择标记，那么就不会画出标记。如果选择了任何一种标记，那么就会在每个数据点的位置画出所选择的标记符号，但是不会用直线连接这些数据点，除非同时还声明了所选择的线型。

如果字符串中包含了颜色、标记和线型，那么就将颜色应用到标记和线条中，为了给标记声明一种不同的颜色，可用不同的声明字符串再绘制一次相同的数据，例如：

```
>> plot(x,y,'b:p',x,z,'c-',x,1.2*z,'m+')
>> title('Figure 26.4: Linestyles and Markers')
```

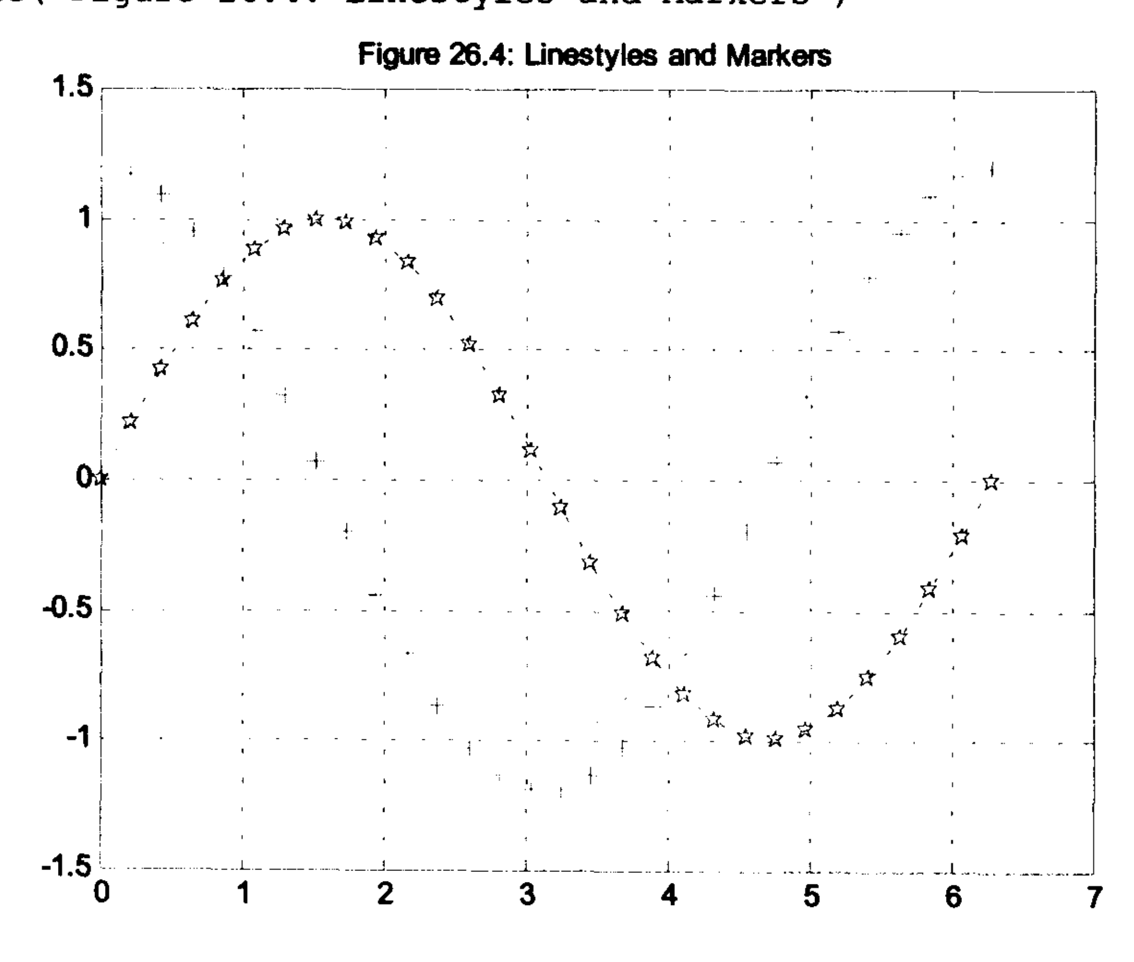


图 26.4 线型和标记

对于本章中的多数图形来说，用户的计算机上都会显示出颜色，但是在这里显示的图形却无法显示出颜色，用户只需要将例子中的命令输入到 Matlab 中，就能看到颜色的效果。

26.3　图形格栅、轴框和标签

命令 grid on 将格栅线在坐标标记线的位置添加到当前的图形中。命令 grid off 将格栅线删除。没有参数的 grid 命令在添加格栅线和删除格栅线之间切换。对于大多数图形，Matlab 默认的设置是 grid off。如果用户想要给所有图形都默认地加上格栅线，那么请将下面这几行语句添加到用户的 startup.m 文件中。

```
set(0, 'DefaultAxesXgrid','on')
set(0, 'DefaultAxesYgrid','on')
set(0, 'DefaultAxesZgrid','on')
```

这几行语句展示了 Matlab 中的图形句柄特性的用法，以及如何设置默认的动作。关于这个问题的更多信息请参见第 30 章。

通常的情况下，二维坐标轴是用实线完全包围的，这些实线被称作一个轴箱(axes box)。这个轴箱可以用 box off 关闭，可以用 box on 恢复这个轴箱。box 命令在轴箱的打开和关闭状态之间实现切换。可以用 xlabel 和 ylabel 函数分别给水平和垂直坐标轴加上标签。title 函数在图形的顶端加上一行文本。考虑下边这个例子：

```
>> x = linspace(0,2*pi,30);
>> y = sin(x);
>> z = cos(x);
>> plot(x,y,x,z)
>> box off                                  % turn off the axes box
>> xlabel('Independent Variable X')         % label horizontal axis
>> ylabel('Dependent Variables Y and Z')    % label vertical axis
>> title('Figure 26.5: Sine and Cosine Curves, No Box') % title
```

用户可以用 text 函数在用户图形的任何指定位置添加一个标签或者任何其他文本字符串。text 的语法为 text(x,y, 'string')，其中(x, y)标明了以图形坐标轴的单位为单位表示的文本字符串的中左边界的坐标。例如，下边这个代码块将文本'sin(x)'放在了 x=2.5，y=0.7 的位置。

```
>> grid on, box on % turn axes box and grid lines on
>> text(2.5,0.7,'sin(x)')
>> title('Figure 26.6: Sine and Cosine Curves, Added Label')
```

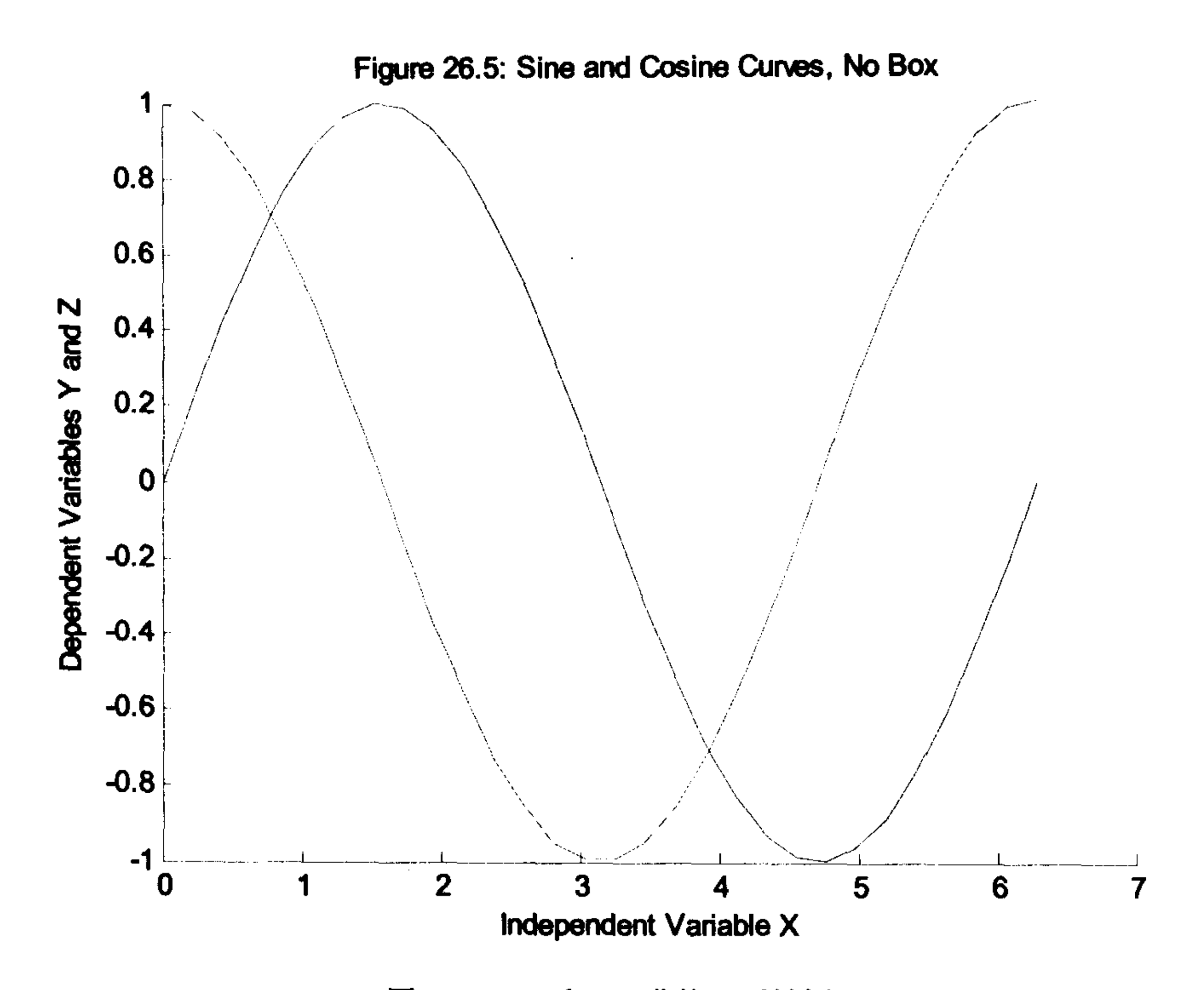


图 26.5 sin 和 cos 曲线，无轴框

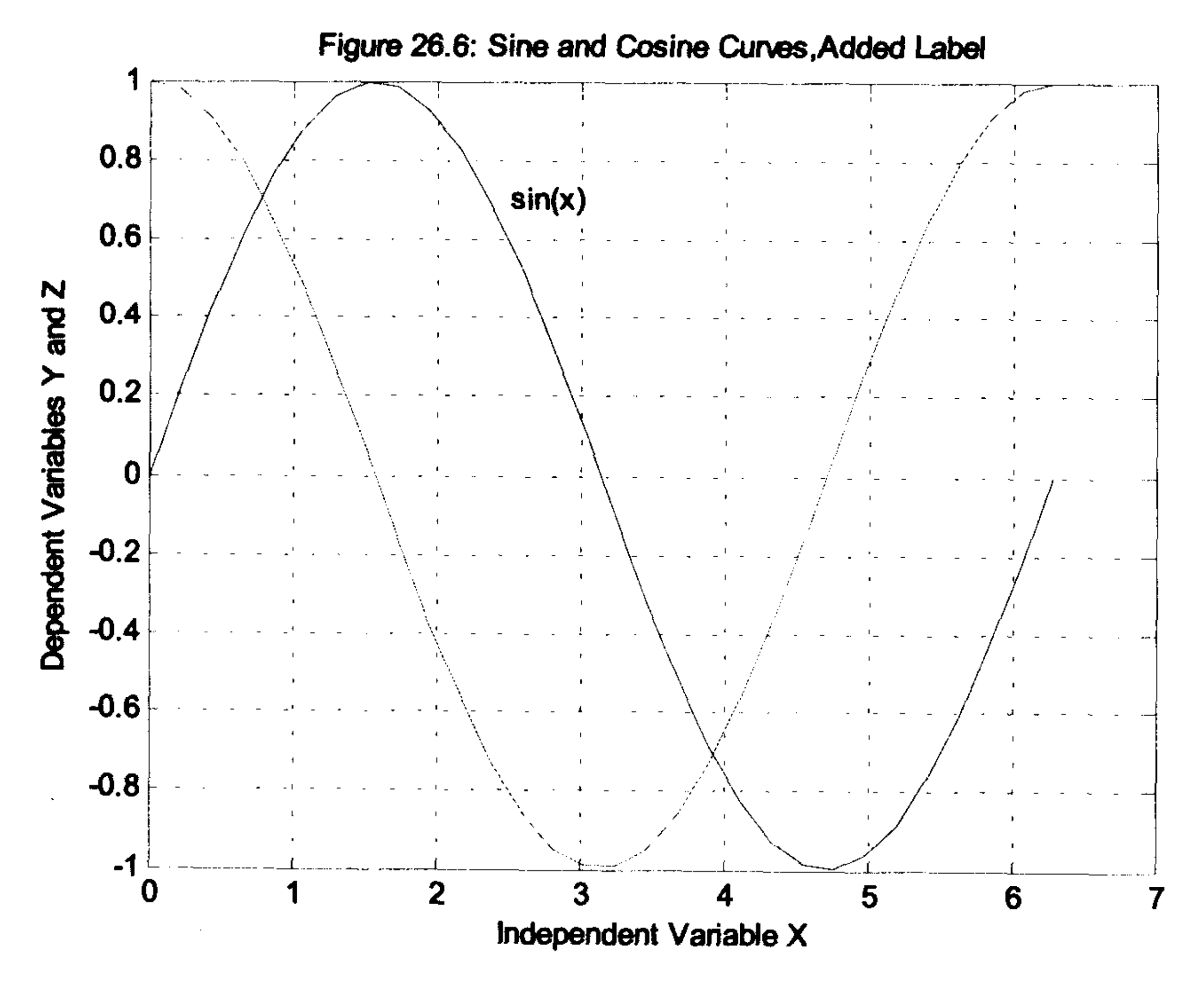


图 26.6 sin 和 cos 曲线，加标签

如果用户想要添加一个标签，但是又不想停下来给出使用的坐标，可以用鼠标来放置一个文本字符串。函数 gtext('text')切换到当前的图形窗口，并给出一个随着鼠标移动的十

字图标，然后等待鼠标点击或者按下键盘按键。当用户点击鼠标或者按下键盘按键的时候，gtext 的字符串参数就被放置在这个时候鼠标所在的位置上，这个字符串的左下角所在的位置就是刚才鼠标所在的位置。

26.4　定制图形坐标轴

Matlab 利用命令 axis，使得用户可以完全地控制用户图形的水平和垂直坐标轴的刻度和外观。axis 命令的特性很多，这里我们只讲述最有用的几个。下表给出了 axis 最主要的特性：

命令	描述
axis([xmin xmax ymin ymax])	设置当前图形的坐标范围
V = axis	返回包含当前坐标范围的一个行向量
axis auto	将坐标轴刻度恢复为自动的默认设置
axis manual	冻结坐标轴刻度，此时如果 hold 被设定为 on，那么后边的图形将使用与前面相同的坐标轴刻度范围
axis tight	将坐标范围设定为被绘制的数据范围
axis fill	设置坐标范围和屏幕高宽比，使得坐标轴可以包含整个绘制的区域。该选项只在 PlotBoxAspectRatio 或 DataAspectRatioMode 被设置为'manual'模式时才有效
axis ij	将坐标轴设置为矩阵模式。此时水平坐标轴从左到右取值，垂直坐标轴从上到下取值
axis xy	将坐标轴设置为笛卡尔模式。此时水平坐标轴从左到右取值，垂直坐标轴从下到上取值
axis equal	设置屏幕高宽比，使得每个坐标轴的具有均匀的刻度间隔
axis image	设置坐标范围，使其与被显示的图形相适应
axis square	将坐标轴框设置为正方形
axis normal	将当前的坐标轴框恢复为全尺寸，并将单位刻度的所有限制取消
axis vis3d	冻结屏幕高宽比，使得一个三维对象的旋转不会改变坐标轴的刻度显示
axis off	关闭所有的坐标轴标签、刻度和背景
axis on	打开所有的坐标轴标签、刻度和背景

可以同时给出 axis 的多个命令。例如，axis auto on xy 是默认的坐标轴刻度。这个 axis 命令只影响当前的图形。因此，它是在 plot 命令之后输入的，就像 grid、xlabel、ylabel、title 和 text 等命令一样，都是在 plot 已经在屏幕上显示了之后输入的命令。请看下边这个例子：

```
x = linspace(0,2*pi,30);
y = sin(x);
plot(x,y)
title('Figure 26.7: Fixed Axis Scaling')
axis([0 2*pi -1.5 2])      % change axis limits
```

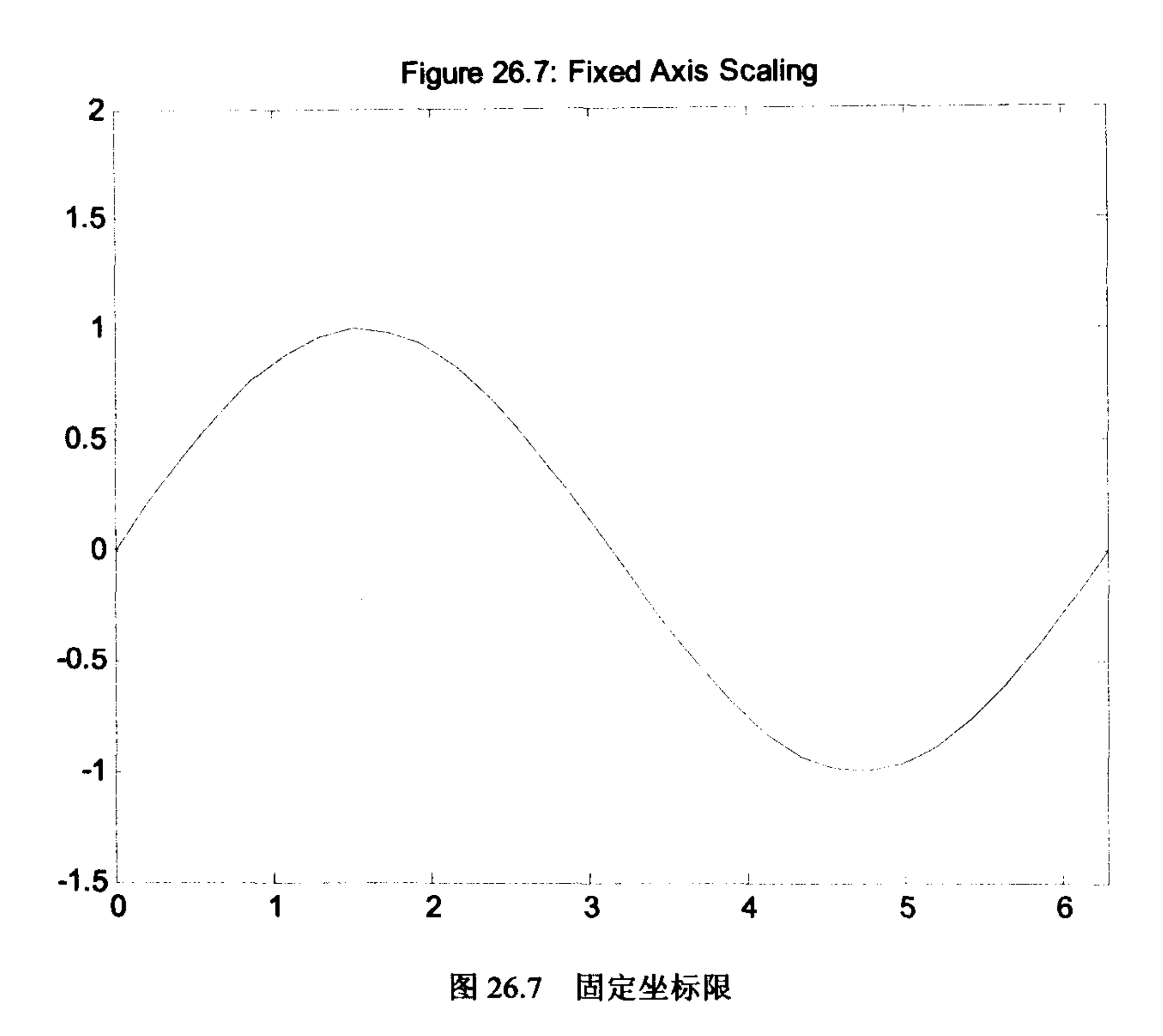


图 26.7 固定坐标限

请注意，通过将 x 轴的最大值设定为 2*pi，这个图形的坐标轴就在 2*pi 的地方结束，而没有将这个坐标限向上进位到 7。弄清楚不同的 axis 命令参数都实现什么功能的最简单方法就是在 Matlab 中生成一个像上边那样简单的图形，然后输入多个 axis 命令，就能够看到图形发生了什么变化。

当用户仅仅想改变一个坐标轴的坐标限的时候，axis 命令就显得很麻烦，因为它要求用户输入所有坐标轴的坐标限。为了解决这个问题，Matlab 提供了函数 xlim、ylim 和 zlim，这些命令都在 xlim 的帮助文本中得到了描述，对 ylim 和 zlim 而言，只需将这个文本中的相应地方改成 ylim 和 zlim 就可以了。

```
>> help xlim
XLIM X limits.
   XL = XLIM              gets the x limits of the current axes.
   XLIM([XMIN XMAX])      sets the x limits.
   XLMODE = XLIM('mode')  gets the x limits mode.
   XLIM(mode)             sets the x limits mode.
                          (mode can be 'auto' or 'manual')
   XLIM(AX,...)           uses axes AX instead of current axes.
   XLIM sets or gets the XLim or XLimMode property of an axes.
   See also pbaspect, daspect, ylim, zlim.
```

26.5 多个图形

用户可以用 hold 命令在一个已经存在的图形上添加一个新的图形。当用户输入 hold on 命令的时候，在用户输入新的 plot 函数的时候，Matlab 不会将现存的坐标轴删除。相反，

它将新的曲线添加到当前的坐标轴中。但是，如果新的数据超出了当前坐标限的范围，Matlab 就将坐标轴重新刻度。输入 hold off 命令就将当前的图形窗口中的图形释放，用以绘制新的图形。不带参数的 hold 命令实现 hold 设置的切换，例如：

```
>> x = linspace(0,2*pi,30);
>> y = sin(x);
>> z = cos(x);
>> plot(x,y)
>> hold on
>> ishold % return 1 (True) if hold is ON
ans =
    1
>> plot(x,z,'m')
>> hold off
>> ishold % hold is no longer ON
ans =
    0
>> title 'Figure 26.8: Use of hold command'
```

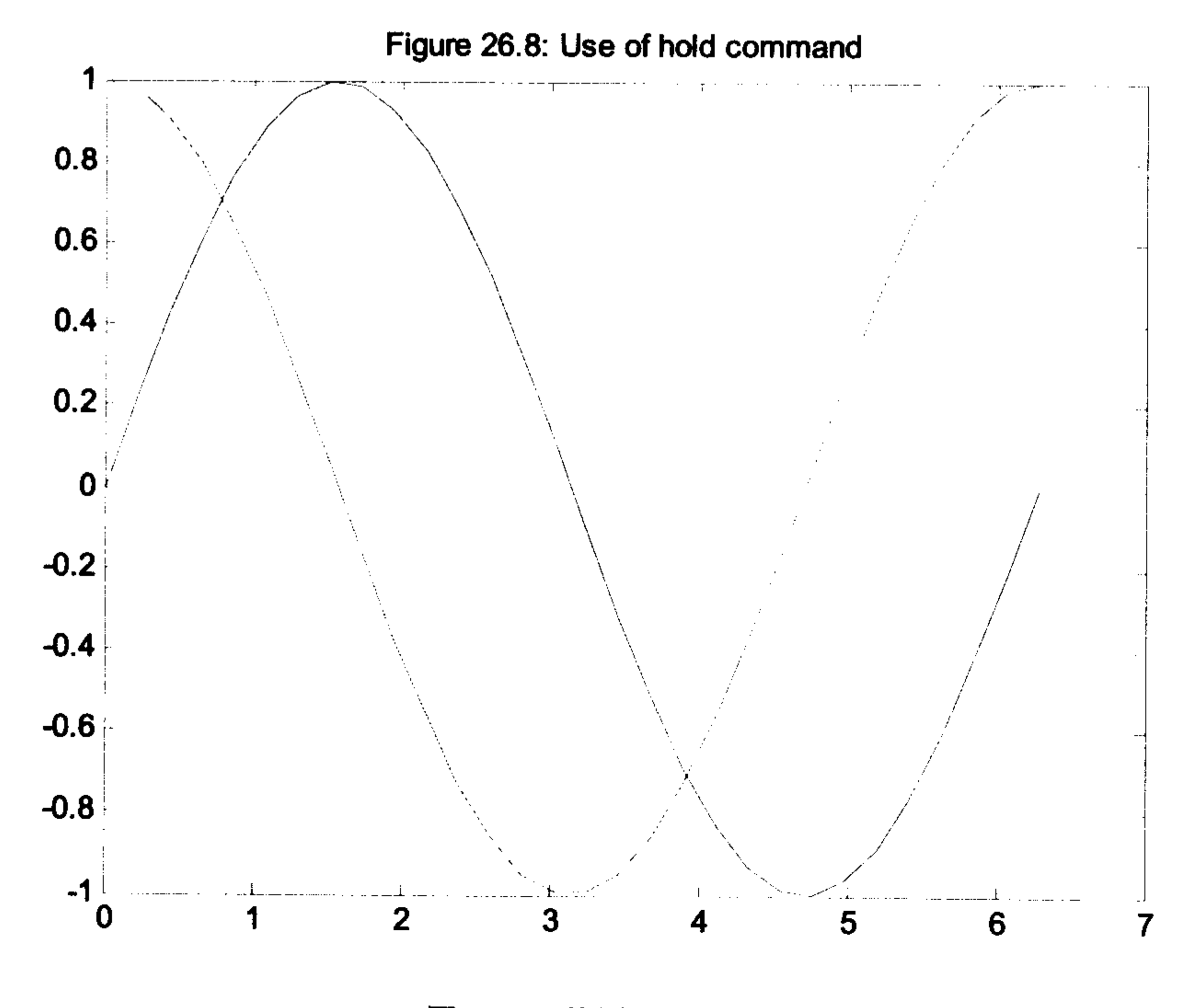


图 26.8 使用 hold 命令

请注意，这个例子声明了第二条曲线的颜色。因为在每个 plot 函数中只有一组数据数组，因此如果不特别声明，每个 plot 函数线条的颜色都会被设置成颜色列表中的第一种颜色，这样就使得图中的两条曲线有相同的颜色。还需要注意的是，标题文本没有包含在括号中，但是效果和在括号中一样。在上边的这种 title 的使用格式中，title 被解释成一个命令，而不是一个函数。

26.6 多个图形窗口

用户可以生成多个图形窗口，并且将不同的数据以不同的方式画在各个窗口中。为了生成一个新的图形窗口，用户需要在命令窗口中使用 figure 命令，或者选择命令窗口或图形窗口 File 菜单中的 New Figure 菜单项。用户可以通过用鼠标点击窗口或者使用 figure(n) 命令将一个特定的图形窗口选定为被激活的或者是当前的图形窗口，其中 n 是这个窗口的编号。当前的图形窗口就是为应用后续的绘图函数而被激活的窗口。

每当生成一个新的图形窗口的时候，就用一个数字来标记它，也就是说，它的句柄被返回并且存储下来以便将来使用。这个图形句柄还被显示在图形窗口的标题栏中。在生成一个新的图形窗口的时候，它被显示在屏幕上的默认的图形窗口位置。这样，当多个图形窗口被生成的时候，每个新的窗口就覆盖了上一个窗口。为了同时看见所有的窗口，用户只需将鼠标指针置于图形窗口的标题栏上再拖动这些窗口。

为了在一个已有的图形窗口中绘制一个新的图形，这个窗口必须是被激活的，或者是当前的图形窗口。在要选定的图形窗口上单击就可以使这个窗口成为当前窗口。在 Matlab 中，输入 figure(h)命令，其中 h 是图形句柄，就使得相应的图形窗口成为被激活的或者当前的窗口。只有当前的图形窗口才对 axis、hold、xlabel、ylabel、title 和 grid 命令作出响应。

图形窗口可以用鼠标关闭窗口的方式来删除，关闭的方式就和在计算机上关闭一个窗口的方式一样。另外，还可以使用命令 close 来关闭窗口。例如：

```
>> close
```

这条命令将当前的图形窗口关闭。

```
>> close(h)
```

这条命令将句柄为 h 的图形窗口关闭。

```
>> close all
```

这条命令将所有的图形窗口关闭。

如果用户只是想将一个图形窗口的内容擦除而不关闭它，就用命令 clf。例如：

```
>> clf
```

这条命令将当前的图形窗口擦除。

```
>> clf reset
```

这条命令将当前的图形窗口擦除，然后将诸如 hold 这样的所有属性重新设置为它们的默认状态。

26.7 子图

一个图形窗口可以包含多套坐标轴系。命令 subplot(m,n,p)将当前的图形窗口分成一个维数为 m×n 的绘图区域数组，并且将第 p 个绘图区域选定为当前的绘图区域。子图是从

最上边一行开始从左到右，然后第二行从左到右，一直到最后一行进行编号的，例如：

```
x = linspace(0,2*pi,30);
y = sin(x);
z = cos(x);
a = 2*sin(x).*cos(x);
b = sin(x)./(cos(x)+eps);

subplot(2,2,1)  % pick the upper left of a 2-by-2 grid of subplots
plot(x,y),axis([0 2*pi -1 1]), title('Figure 26.9a: sin(x)')

subplot(2,2,2)  % pick the upper right of the 4 subplots
plot(x,z),axis([0 2*pi -1 1]),title('Figure 26.9b: cos(x)')

subplot(2,2,3)  % pick the lower left of the 4 subplots
plot(x,a),axis([0 2*pi -1 1]),title('Figure 26.9c: 2sin(x)cos(x)')

subplot(2,2,4)  % pick the lower right of the 4 subplots
plot(x,b),axis([0 2*pi -20 20]), title('Figure 26.9d: sin(x)/cos(x)')
```

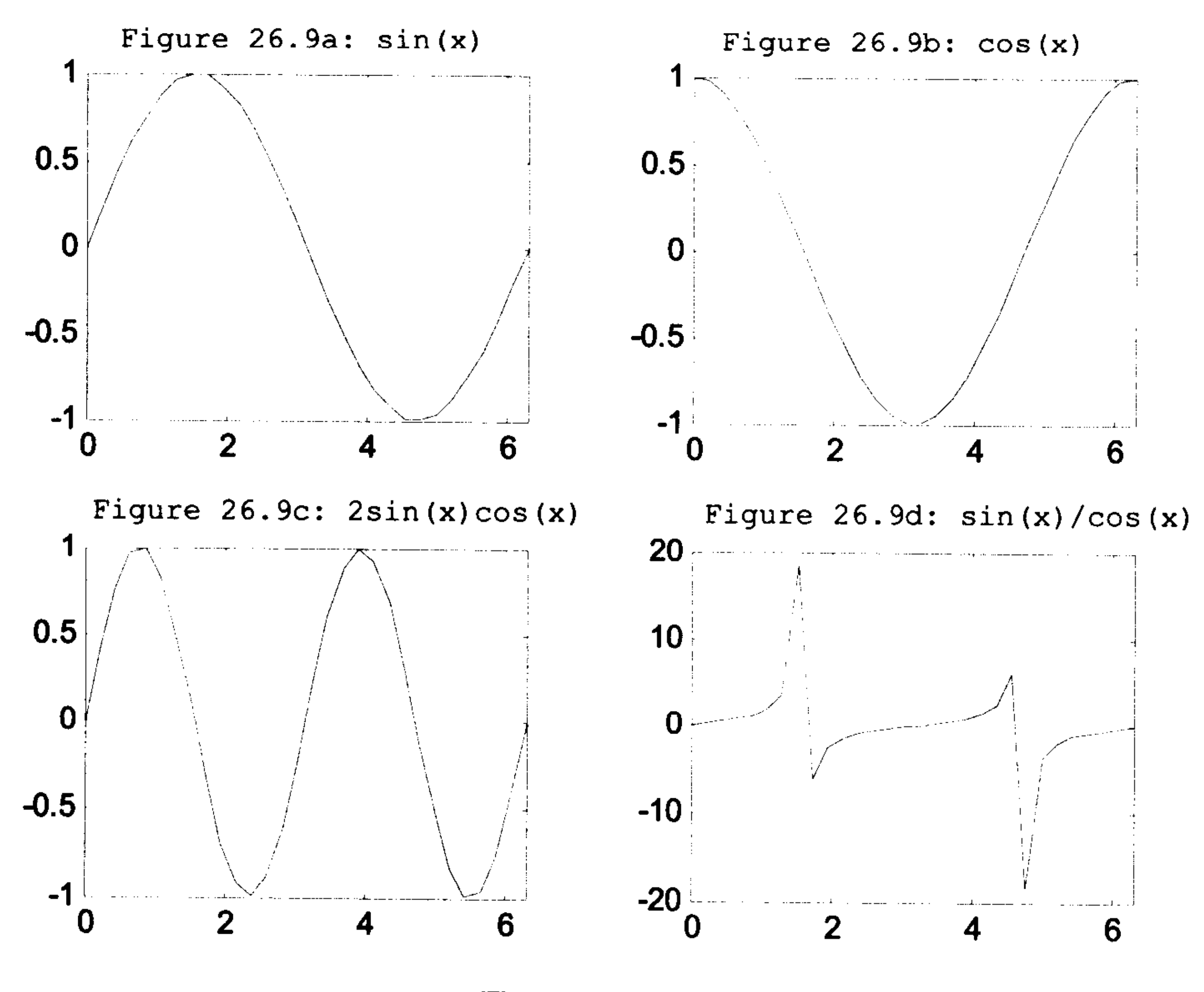


图 26.9 绘制子图

请注意当一个特定的子图被激活的时候，它是惟一的一个对命令 axis、hold、xlabel、ylabel、title、grid 和 box 进行响应的子图。其他子图不受任何影响。另外，被激活的子图在用户输入另一个 subplot 或者 figure 命令之前会一直保持被激活状态。当一个新的 subplot 命令改变了在图形窗口中的子图数量的时候，原来的那些子图就被擦除掉了，以便给新的子图腾出空间。为了回到默认的模式并且在整个图形窗口中只用一套坐标轴，可以使用命令 subplot(1,1,1)。当用户打印一个包含了多个子图的图形窗口的时候，所有这些子图都被

打印在相同的一页上。例如，若当前的图形窗口包含了 4 个子图，并且排列方向模式为横向模式，那么每个子图就使用打印的一页的四分之一。

26.8 交互式画图工具

在图形窗口菜单栏和工具栏出现之前，Matlab 提供了几个函数来交互地注释图形。这些函数在本节得到了描述，并且这些函数中的大多数都可以通过图形菜单栏和工具栏来得到。

可以用图例来标志用户图形中的数据集，而不是用单个的文本字符串来标志。命令 legend 就在图形上生成了一个图例框，使得用户可以输入用户在图上的任何一行所提供的任何文本。如果用户想要移动这个图例框，只需要将鼠标指针置于图例框左下角附近的位置，然后就可以将这个图例框拖动到想到的位置。legend off 将图例框删除。请看下边这个例子：

```
>> close % close figure containing subplots
>> x = linspace(0,2*pi,30);
>> y = sin(x);
>> z = cos(x);
>> plot(x,y,x,z)
>> legend('sin(x)','cos(x)')
>> title('Figure 26.10: Legend Example')
```

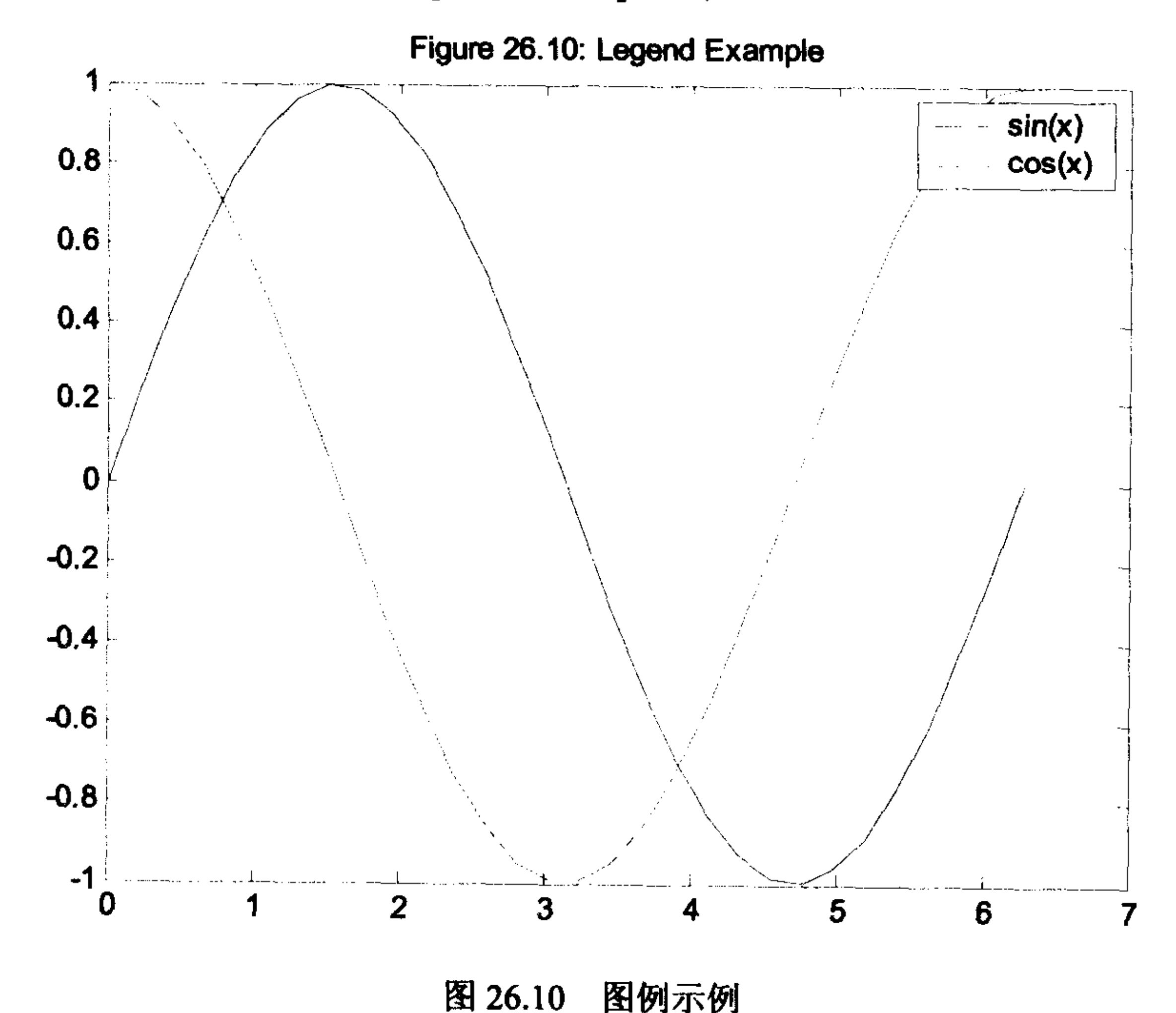


图 26.10 图例示例

Matlab 提供了一个交互的工具来将二维图形的局部进行放大，以便能够看得更详细，或者将一个关心的区域放大。命令 zoom on 将放大模式打开。在图形窗口中单击鼠标，就将图形以 2 为倍数，以鼠标点击的点为圆心进行放大。用户每单击一次，图形就放大一次。右击鼠标就将图形以 2 为倍数缩小。用户还可以单击并拖动鼠标来将形成的特定方形区域

进行放大。命令 zoom(n)以 n 为倍数进行放大。zoom out 将图形返回到它的初始状态。zoom off 关闭放大模式。不带参数的 zoom 实现当前的图形窗口放大模式打开与关闭之间的切换。图形工具栏和图形窗口菜单还提供了实现这些特性的 GUI 方法。

在某些情况下，从一个图形窗口中的图形中选择坐标点是很方便的。在 Matlab 中，这个特性是通过 ginput 函数实现的。ginput 的如下调用形式[x,y]=ginput(x)就从当前的图形或者子图中获取 n 个点，具体是哪些点要依赖于鼠标在这个图形或者子图中点击的位置。如果用户在所有的 n 个点被选取完之前输入了回车键，那么 ginput 就中断执行过程，即使所得到的数据点不足 n 个。在向量 x 和 y 中所返回的数据分别是所选择点的 x 坐标和 y 坐标值。返回的数据不需要一定是来自绘制图形的点，它可以是鼠标点击的位置的显式 x 坐标和 y 坐标值。如果在图形或者子图的坐标限之外选择了数据点，例如，在图形框外，那么返回的数据点就是推断出来的值。

当这个函数被用在一个包含了子图的图形窗口中的时候，可能在某种程度上让人感到困惑。返回的数据点是当前或者被激活的子图的。这样，如果 ginput 是在一个 subplot(2,2,3)命令之后输入的，那么返回的数据就是 subplot(2,2,3)中的第三个子图中的数据点。如果数据点是从其他子图获得的，那么这些数据仍旧是以 subplot(2,2,3)中的坐标轴为基准的。当用户希望获得没有声明数量的数据点的时候，可以用[x,y]=ginput 这样的不带输入参数的命令格式。这里，用户可以不断地收集数据点，直到按回车键为止。本章早些时候描述的 gtext 函数使用了函数 ginput 以及 text 函数来将文本放置在鼠标点击的位置。

26.9 屏幕刷新

由于屏幕刷新相对而言很消耗时间，因此 Matlab 并不总是在每个图形命令之后都刷新屏幕。我们举一个简单的例子，例如，如果用户在 Matlab 提示符下逐个输入下面的命令，Matlab 就会在执行每个命令（包括 plot、axis 和 grid）时刷新屏幕：

```
>> x = linspace(0,2*pi); y = sin(x);
>> plot(x,y)
>> axis([0 2*pi -1.2 1.2])
>> grid
```

但如果用户将这些命令在同一行上输入，例如：

```
>> plot(x,y), axis([0 2*pi -1.2 1.2]), grid
```

那么 Matlab 只对屏幕进行一次刷新。另外，如果上述命令出现在一个 M 脚本文件或 M 函数文件中，Matlab 也只进行一次屏幕刷新。

总的来说，在 Matlab 7 中，以下 6 种情况可以导致屏幕刷新：

（1）在命令窗口中返回到下一个 Matlab 提示符时，即用户在输入命令后按回车键。
（2）遇到一个临时中止执行的函数，比如 pause、keyboard、input 和 waitforbuttonpress。
（3）执行一个 getframe 命令。
（4）执行一个 drawnow 命令。
（5）执行一个 figure 命令。

（6）重新设置一个图形窗口的大小。

在上述 6 种情况中，drawnow 命令可使用户在任何时候强制 Matlab 刷新屏幕。

26.10 特殊的二维图形

在前面几节，我们主要针对 plot 函数讲述了均匀坐标轴刻度下的绘图。但在很多情况下，仅在均匀坐标轴刻度下绘制曲线和数据点并不能完全满足运算需要。因此，Matlab 还提供了其他一些基本二维绘图函数，以及一些特殊的绘图函数（这些函数以 M 函数文件的形式给出）。

为了可以使用对数坐标轴绘制图形，Matlab 提供了函数 semilogx，用来将 x 轴转化为对数刻度；函数 semilogy，用来将 y 轴转化为对数刻度；函数 loglog，用来将 x-y 两个轴转化为对数刻度。这三个函数的基本用法与 plot 函数完全一致。

Matlab 还提供了 area 函数用于构建一个层叠的区域图。对于给定的向量 x 和 y，area(x,y) 和 plot(x,y)将绘制相同的图形，只不过 area(x,y)将在所绘制的曲线下面填充颜色。用户可以设置填充区的下限，如果没有设置，默认值为 0。要实现区域层叠，可使用 area(X,Y)的调用格式，其中 Y 是一个矩阵，X 可以是一个矩阵，也可以是一个长度等于 Y 的行数的向量。如果省略了 X，area 就将 X 默认为 X=1:size(Y,1)。下面给出了一个绘制层叠图形的例子，其中在 area 的调用中省略了 X：

```
>> z = -pi:pi/5:pi;
>> area([sin(z);cos(z)])
>> title('Figure 26.11: Stacked Area Plot')
```

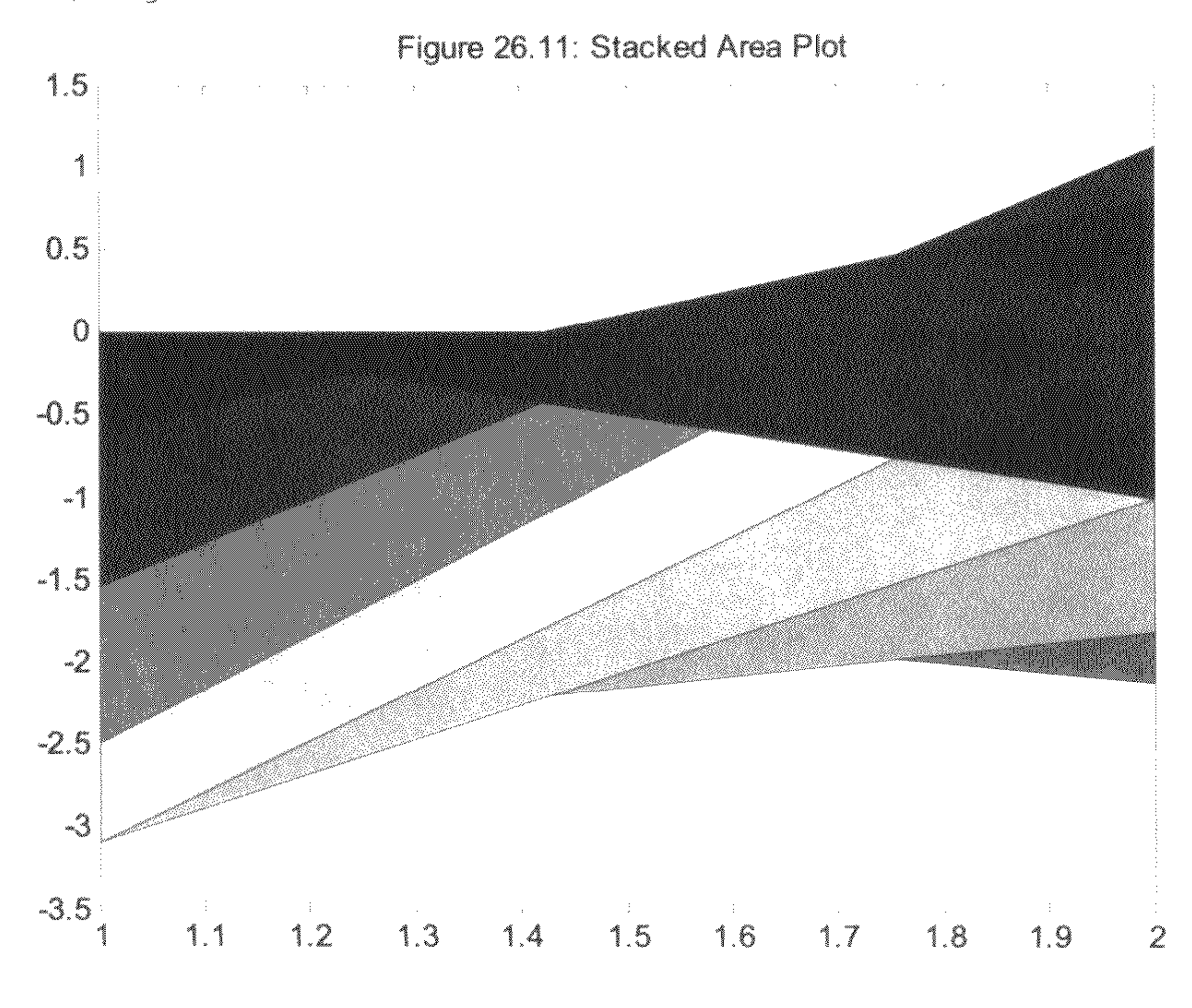


图 26.11 层叠区域绘制

Matlab 提供了 fill 函数用于绘制一个多边形并用指定的颜色填充。fill 函数的基本调用方式为：fill(x,y, 'c')，该命令用指定的颜色 c 填充由列向量 x 和 y 定义的二维多边形，其中，数据对(x(i), y(i))用于确定多边形的每个顶点位置。如果(x(i), y(i))中第一个和最后一个数据对不相同，Matlab 就将最后一个点和第一个点连接起来形成一个封闭的多边形。就像 plot 函数一样，fill 可以有任意数量的顶点对和对应的颜色。另外，当 x 和 y 是相同维数的矩阵的时候，x 和 y 的列都被假定为描述了不同的多边形。请看下边这个例子：

```
>> t = (1:2:15)'*pi/8;
>> x = sin(t);
>> y = cos(t);
>> fill(x,y,'r') % a filled red circle using only 8 data points
>> axis square off
>> text(0,0,'STOP',…
   'Color',[1 1 1],…
   'FontSize',80,…
   'FontWeight','bold',…
   'HorizontalAlignment','center')
>> title('Figure 26.12: Stop Sign')
```

这个例子使用了带额外参数的 text(x,y, 'string')函数。参数 Color、FontSize、FontWeight 和 HorizontalAlignment 告诉 Matlab 用句柄图形来修正文本。句柄图形是 Matlab 的基本图形函数的名字。用户可以自己访问这个强大的、功能丰富的图形函数集合。关于这些特性的更多信息请参见第 31 章。

可以用函数 pie(a,b)来生成标准的饼图，其中 a 是一个数值向量而 b 是一个可选的逻辑向量，它描述了一个或者多个需要从这个饼图中拉出的饼片。函数 pie3 使得饼图有三维的显示效果。请看下边这个例子。

```
>> a = [.5 1 1.6 1.2 .8 2.1];
>> pie(a,a==max(a));    % chart a and pull out the biggest slice
>> title('Figure 26.13: Example Pie Chart')
```

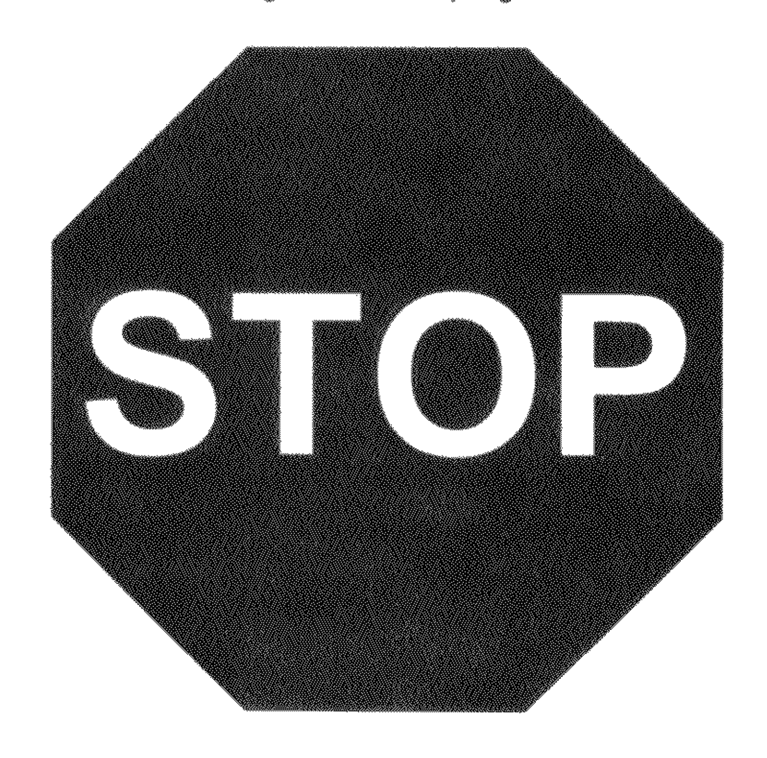


图 26.12　Stop 标志

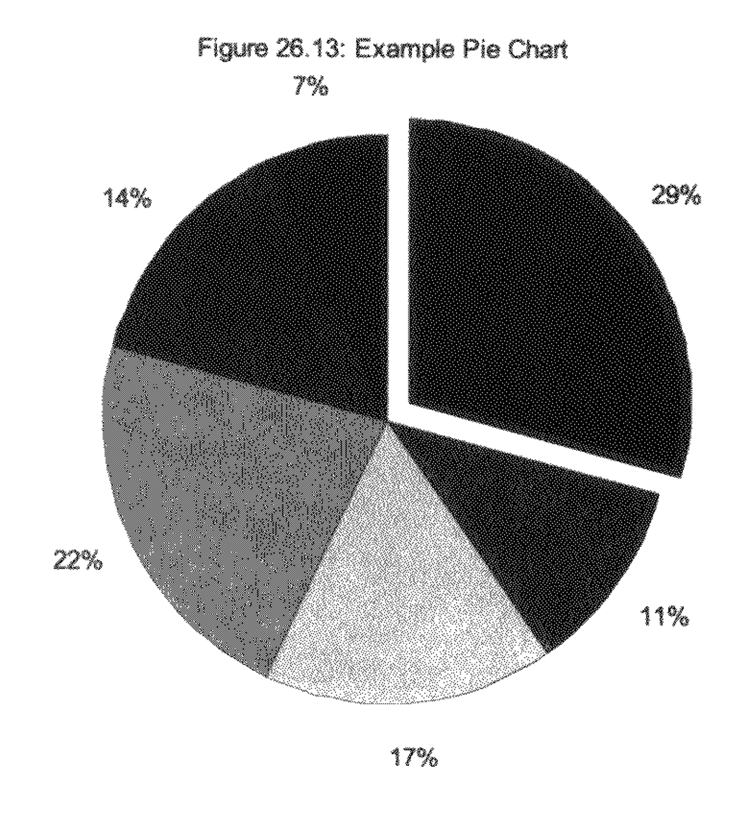
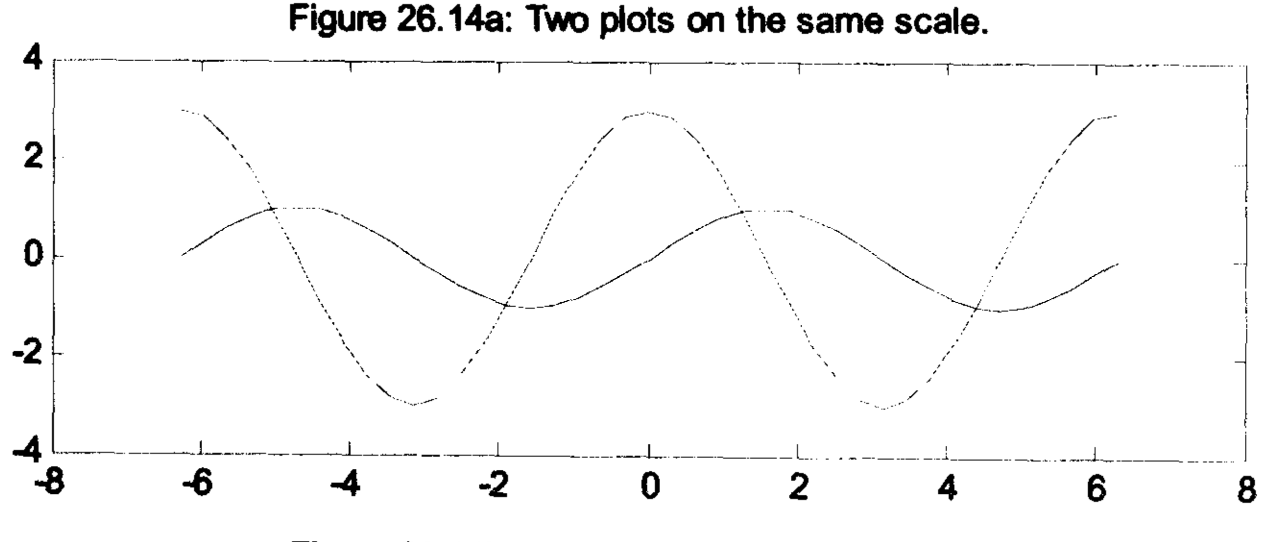


图 26.13　饼图示例

有时候用户希望将两个不同的函数绘制在同一个坐标轴上，但是使用不同的 y 轴刻度。函数 plotyy 就是实现这个功能的，例如：

```
>> x = -2*pi:pi/10:2*pi;
>> y = sin(x);
>> z = 3*cos(x);
>> subplot(2,1,1), plot(x,y,x,z)
>> title('Figure 26.14a: Two plots on the same scale.');
>> subplot(2,1,2), plotyy(x,y,x,z)
>> title('Figure 26.14b: Two plots on different scales.');
```



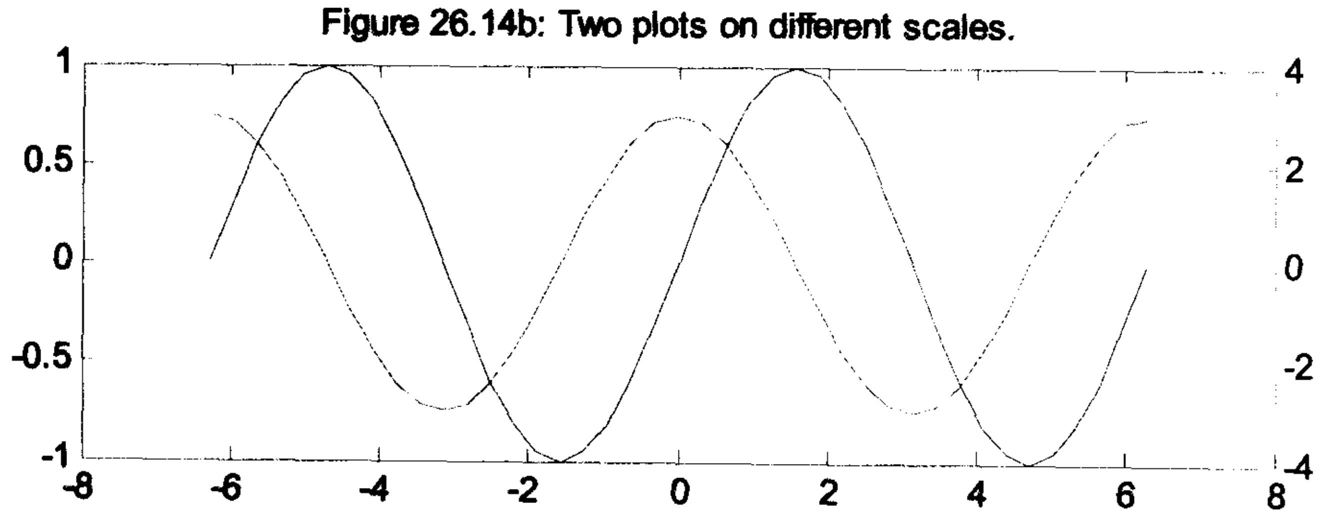


图 26.14 绘制双图时的 y 坐标限选取

条形图和梯形图可以用绘图函数 bar、barh 和 stairs 得到。函数 bar3 和 bar3h 都使得条形图具有三维效果。考虑下边这个例子。

```
>> x = -2.9:0.2:2.9;
>> y = exp(-x.*x);

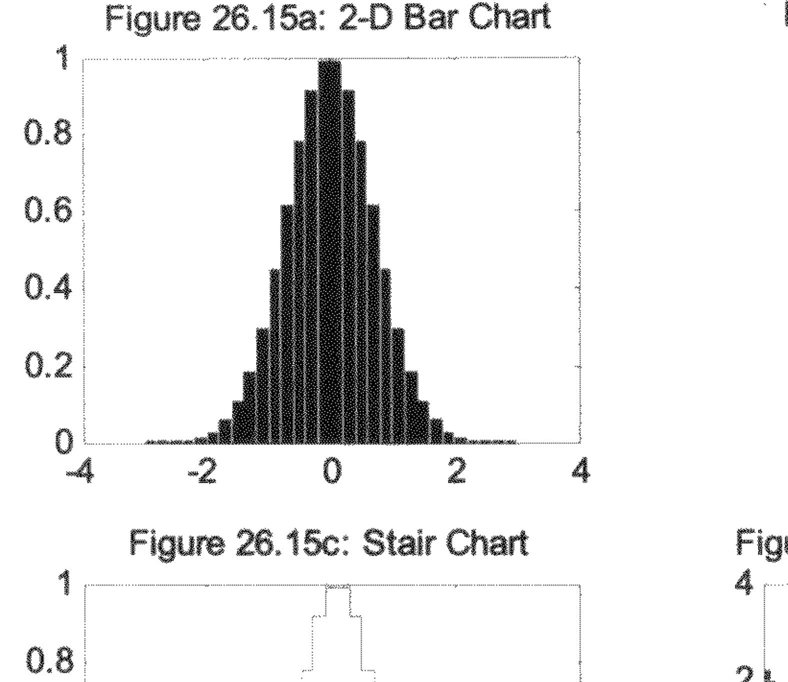
>> subplot(2,2,1)
>> bar(x,y)
>> title('Figure 26.15a: 2-D Bar Chart')

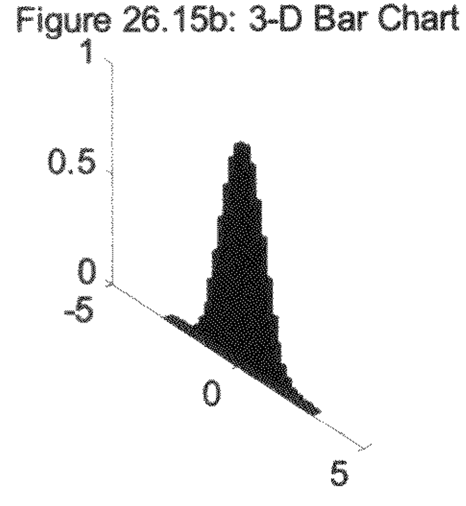
>> subplot(2,2,2)
>> bar3(x,y,'r')
>> title('Figure 26.15b: 3-D Bar Chart')

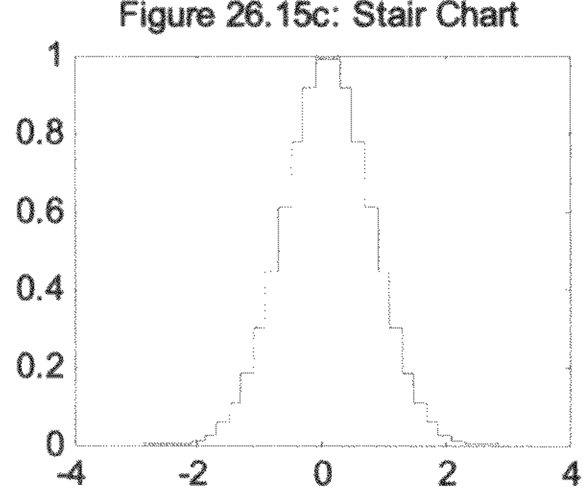
>> subplot(2,2,3)
>> stairs(x,y)
```

```
>> title('Figure 26.15c: Stair Chart')

>> subplot(2,2,4)
>> barh(x,y)
>> title('Figure 26.15d: Horizontal Bar Chart')
```







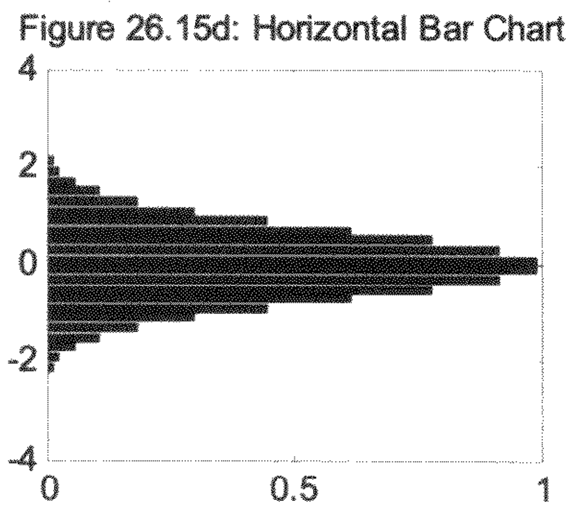


图 26.15 条形图和梯形图

不同的 bar 函数都对所有的条形图接受一个颜色参数。条形图也可以一组一组地生成或者层叠生成。bar(x,Y)这样的调用格式（其中 x 是一个向量，而 Y 是一个矩阵）就根据 Y 的列数画出一组条形图。bar(x,Y,'stacked')就画出一组垂直层叠的条形图。barh、bar3 和 bar3h 都有相同的选择功能。

柱状图展示了一个向量中值的分布情况。hist(y)为向量 y 中的数据画出了一个 10 柱的柱状图。hist(y,n)（其中 n 是一个标量）就画出一个有 n 个柱的柱状图。hist(y,x)（其中 x 是一个向量）就绘制出一个 x 中声明的柱的柱状图，请看下面这个例子：

```
>> x = -2.9:0.2:2.9;          % specify the bins to use
>> y = randn(5000,1);         % generate 5000 random data points

>> hist(y,x)                  % draw the histogram
>> title('Figure 26.16: Histogram of Gaussian Data')
```

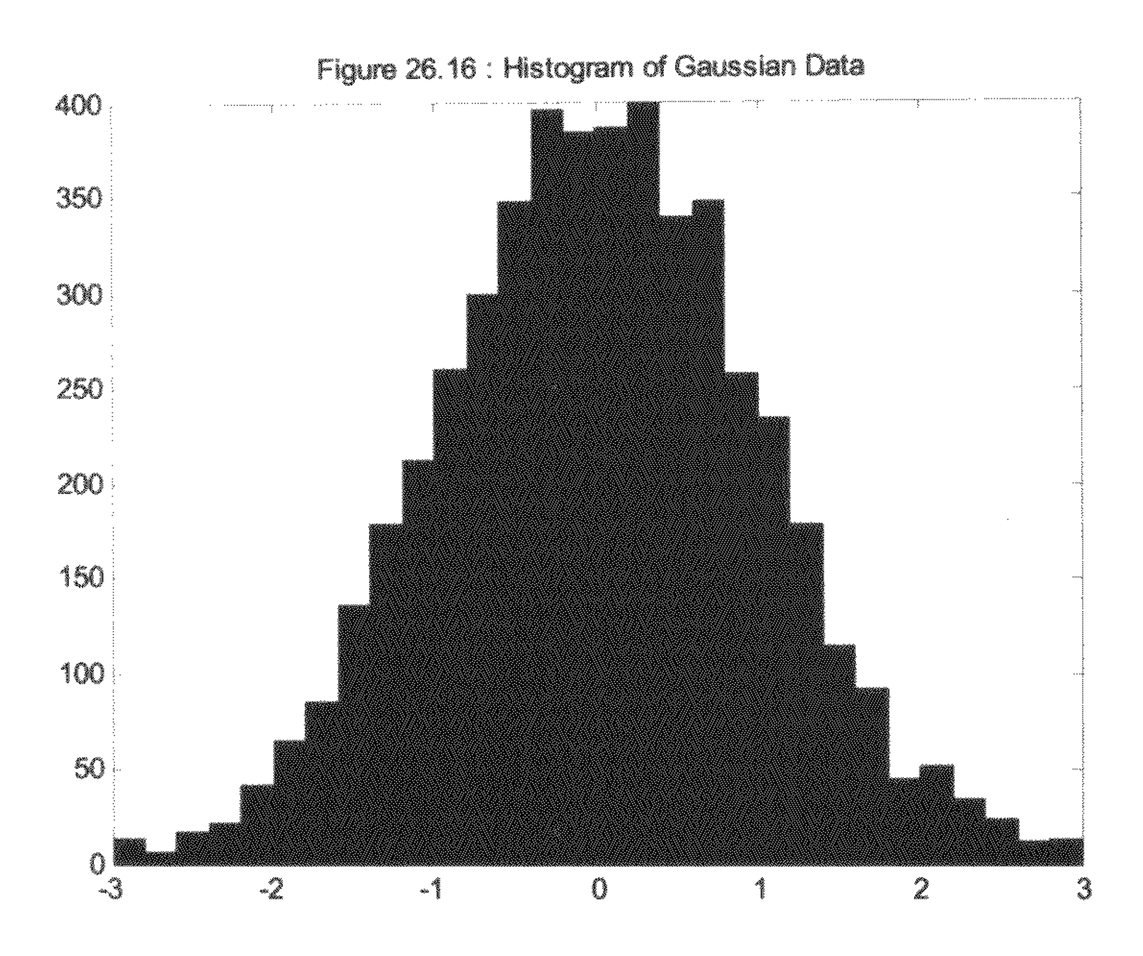


图 26.16 高斯分布数据的直方统计图

离散的数据序列可以用 stem 函数进行绘制。stem(z)生成一个向量 z 中的数据点的图形，其中各个点用一条直线与水平坐标轴相连。可以用一个可选的字符串参数来声明线型。stem(x,z)将 z 中的数据点绘制在 x 值所声明的位置，如下例所示：

```
>> z = randn(30,1);            % create some random data
>> stem(z,'--')                % draw a stem plot using dashed linestyle
>> set(gca,'YGrid','on')       % turn grid on Y-axis only
>> title('Figure 26.17: Stem Plot of Random Data')
```

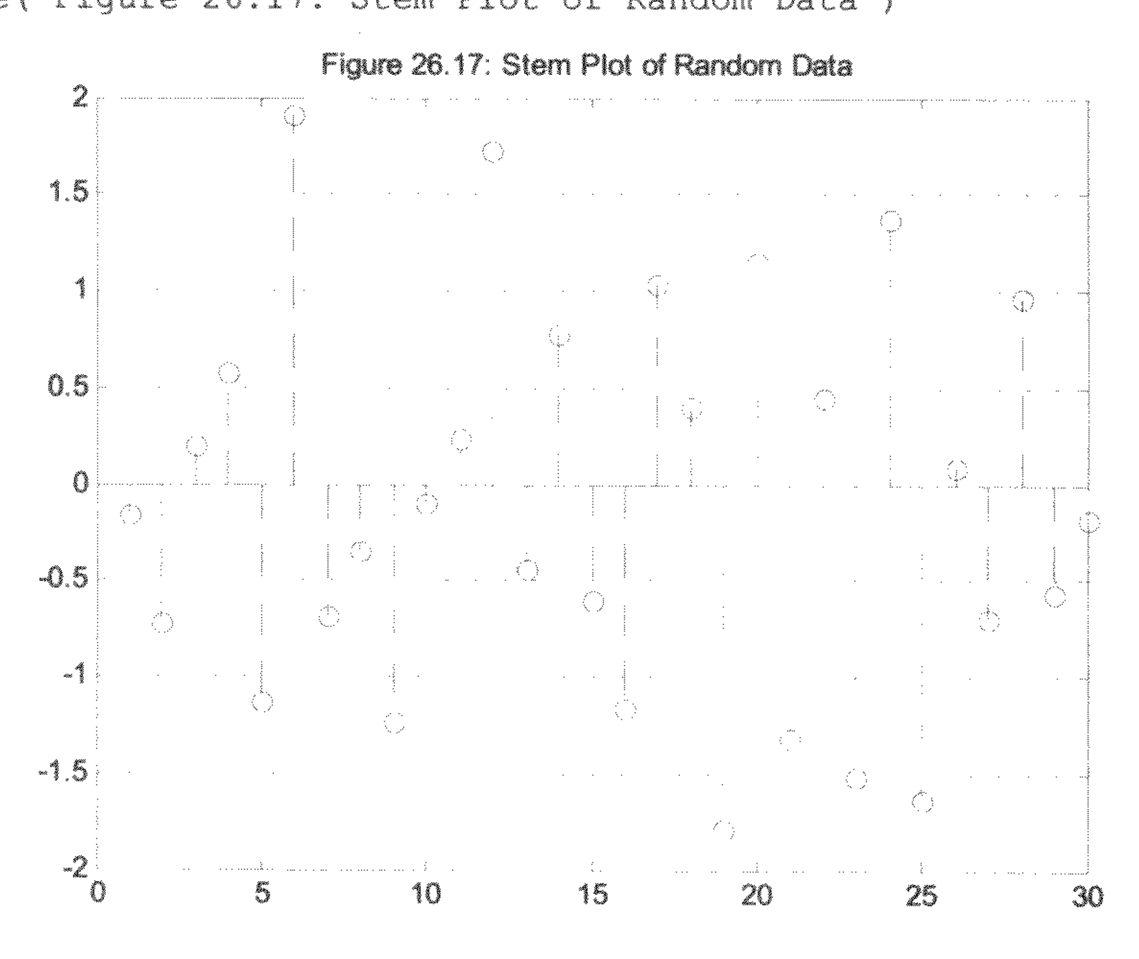


图 26.17 随机分布数据的柄状图

一个图形可以在数据点的位置包含误差线段。errorbar(x,y,e)绘制出向量 x 向量 y 的图形，以及用向量 e 声明的误差线段。所有的向量都必须是相同长度的。对于每一个数据点(x(i),y(i))，一个误差线段被画在在数据点上 e(i)的距离和数据点下 e(i)的距离的地方，例如：

```
>> x = linspace(0,2,21);         % create a vector
>> y = erf(x);                   % y is the error function of x
>> e = rand(size(x))/10;         % e contains random error values

>> errorbar(x,y,e)               % create the plot
>> title('Figure 26.18: Errorbar Plot')
```

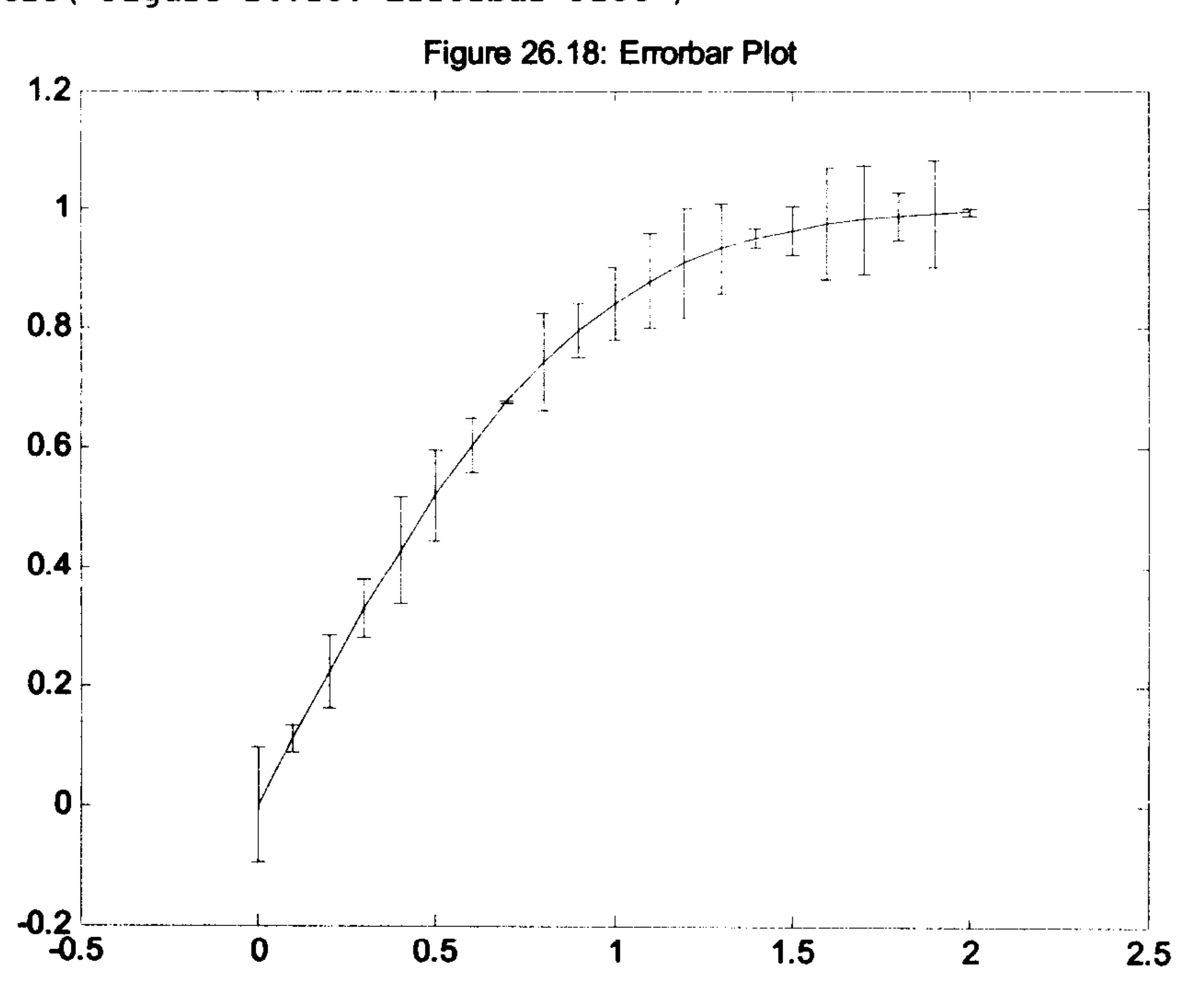


图 26.18 误差线段图

可以用函数 polar(t,r,S)来绘制极坐标中的图形，其中 t 是用弧度表示的角度向量，r 是半径向量，S 是一个可选的字符串，它描述了颜色、符号标记以及/或者线型，例如：

```
>> t = linspace(0,2*pi);
>> r = sin(2*t).*cos(2*t);
>> subplot(2,2,1)
>> polar(t,r),title('Figure 26.19a: Polar Plot')
```

也可以用 compass 和 feather 来绘制复数。compass(z)画出显示了 z 的复数元素角度和幅度的图形，其图形就像从原点发出的箭头一样。feather(z)将同样的这些数据画成从水平直线上的等距离间隔的点上发出的箭头。compass(x,y)和 feather(x,y)等价于 compass(x+i*y)和 feather(x+i*y)。请看下边这个例子：

```
>> z = eig(randn(20));
>> subplot(2,2,2)
>> compass(z)
```

```
>> title('Figure 26.19b: Compass Plot')
>> subplot(2,2,3)
>> feather(z)
>> title('Figure 26.19c: Feather Plot')
```

函数 rose(v)对向量 v 中的角度画出了一幅有 20 柱的极坐标柱状图。rose(v,n)，其中 n 是一个标量，这个函数画出有 n 个柱的柱状图。rose(v,x)，其中 x 是一个向量，这个函数画出一个 x 中声明的柱数的柱状图。请看下边这个例子：

```
>> subplot(2,2,4)
>> v = randn(1000,1)*pi;
>> rose(v)
>> title('Figure 26.19d: Angle Histogram')
```

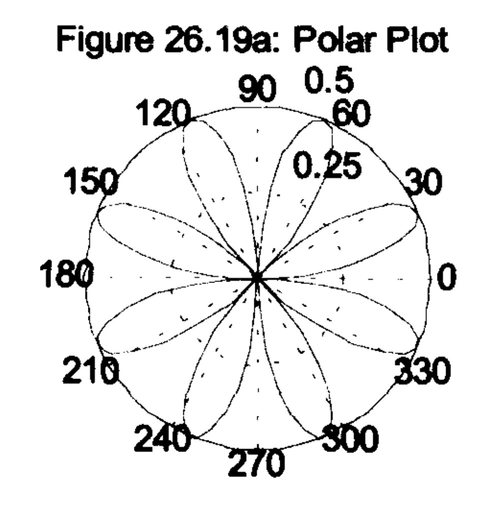
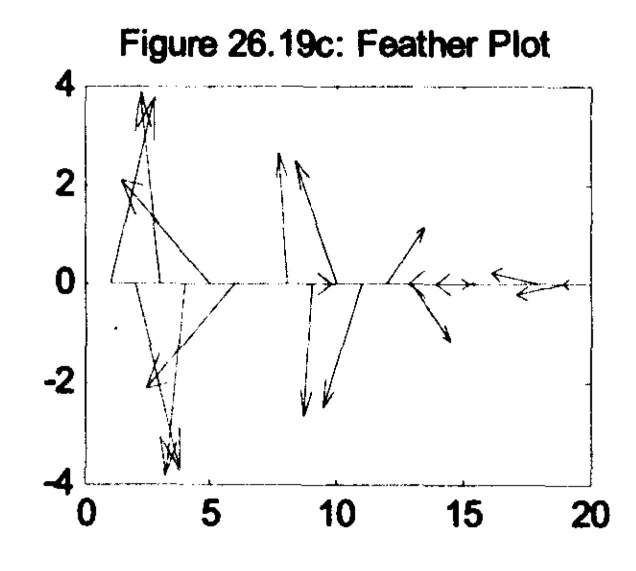


Figure 26.19b: Compass Plot
90 5 120 60 2.5 150 30 180 0 210 330 240 300 270



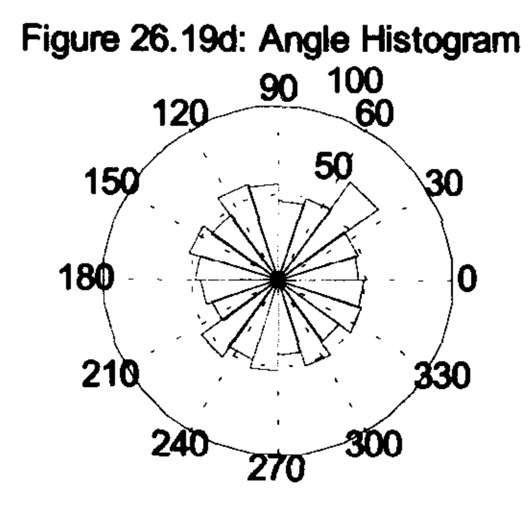


图 26.19 复数的绘制

函数 scatter 生成一个散布图，也就是以各个点为圆心的圆圈图，其中各个圆圈的大小或者颜色可以因点而异。请看下边这个例子：

```
>> x = rand(40,1);
>> y = randn(40,1);
>> area = 20+(1:40);
>> scatter(x,y,area)
>> box on
>> title('Figure 26.20: A scatter plot')
```

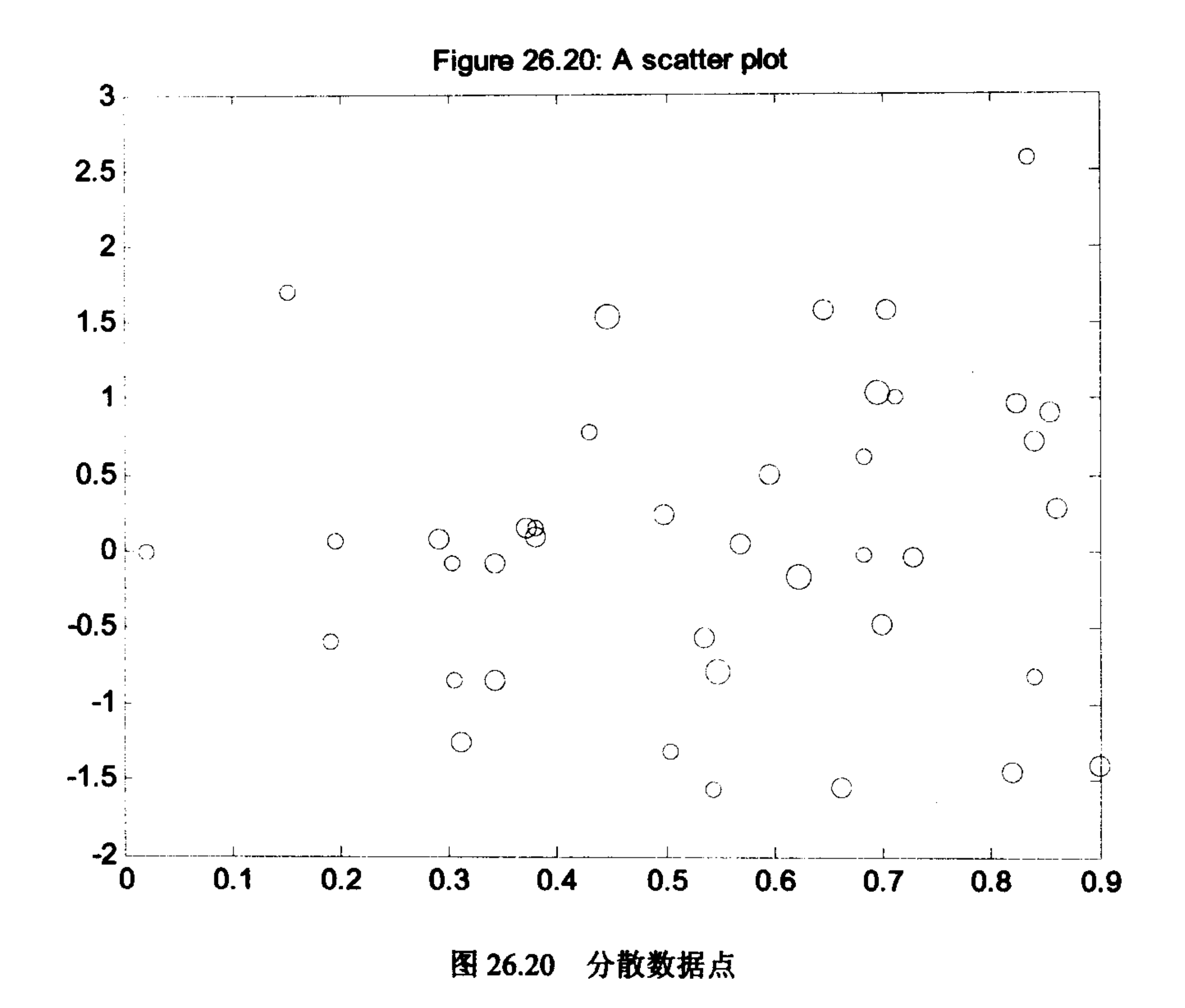


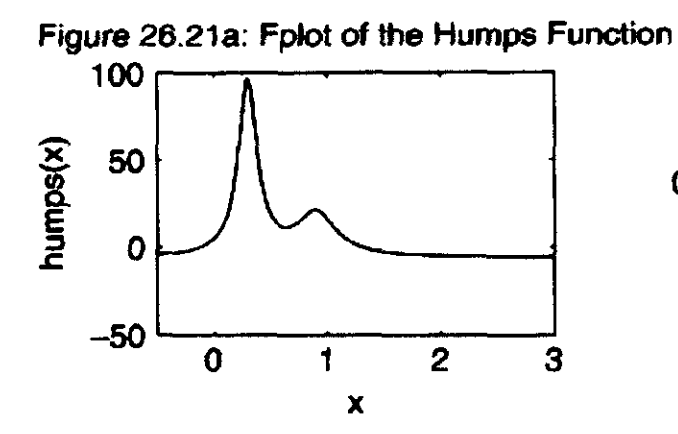
图 26.20　分散数据点

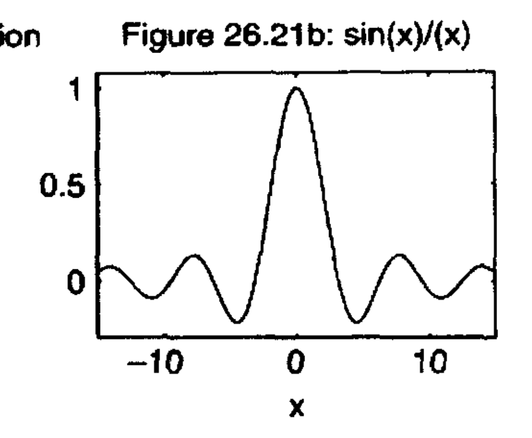
26.11　轻松绘图

当用户不想花费那么多的时间来给一幅图声明数据点的时候，Matlab 提供了函数 fplot、ezplot 和 ezpolar。函数 fplot 绘制由 M 文件名或者函数句柄定义的函数。函数 ezplot 和 ezpolar 绘制由字符串表达式或者符号数学对象定义的函数，两者之间在图形类型上有明显的差别。这些函数仅仅需要用户定义自变量的数据，例如：

```
>> subplot(2,2,1)
>> fplot(@humps,[-.5 3])
>> title('Figure 26.21a: Fplot of the Humps Function')
>> xlabel('x')
>> ylabel('humps(x)')
>> subplot(2,2,2)
>> f_hdl=@(x) sin(x)/(x);
>> ezplot(f_hdl,[-15,15])
>> title(['Figure 26.21b: ' sin(x)/x])

>> subplot(2,2,3)
>> istr = '(x-2)^2/(2^2) + (y+1)^2/(3^2) -1';
>> ezplot(istr,[-2 6 -5 3])
>> axis square
>> grid
>> title(['Figure 26.21c: ' istr])
```





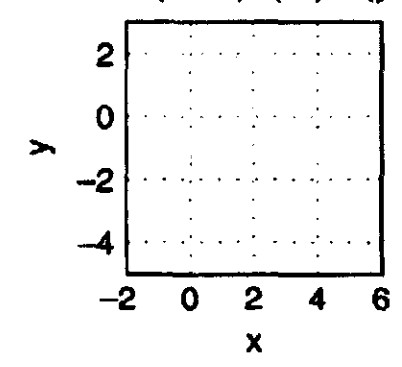


图 26.21 使用 ezplot 函数

最后的这个例子表明，ezplot 可以用来绘制隐含函数。在这个例子中，字符串表达式表示的是一个以(2,-1)为中心的椭圆。

26.12 文本格式

在任何一个字符串中都可以使用多行文本，包括标题和坐标轴标签，以及 text 和 gtext 函数中的输入参数。对于多行文本，只需要使用字符串数组或者单元数组。例如：

```
>> xlabel({'This is the first line','and this is the second.'});
```

这条命令用两行文本给了 x 轴一个标签。请注意，字符串分隔符可以是一个空格、一个逗号或者是一个分号；无论是哪一种分隔符都会得到相同的结果。关于字符串单元数组的更详细信息请参看第 9 章。

通过在字符串中内嵌一个 TeX 命令的子集，在 Matlab 文本字符串中包含的符号数就可以超过 75 个，包括希腊字母和其他特殊字符。下表列出了可以提供的符号和用来定义这些符号的字符串。这些信息还可以通过浏览在线文档中的 text 句柄图形对象的 string 属性得到。

字符串	符号	字符串	符号	字符串	符号
\alpha	α	\rfloor	⌋	\cong	≅
\beta	β	\lfloor	⌊	\approx	≈
\gamma	γ	\perp	⊥	\Re	ℜ
\delta	δ	\wedge	∧	\oplus	⊕
\epsilon	ε	\rceil	⌉	\cup	∪

（续表）

字符串	符号	字符串	符号	字符串	符号
\zeta	ζ	\vee	∨	\subseteq	⊆
\eta	η	\langle	<	\in	∈
\theta	θ	\upsilon	υ	\lceil	⌈
\vartheta	ϑ	\phi	φ	\cdot	·
\iota	ι	\chi	χ	\neg	¬
\kappa	κ	\psi	ψ	\times	×
\lambda	λ	\omega	ω	\surd	√
\mu	μ	\Gamma	Γ	\varpi	ϖ
\nu	ν	\Delta	Δ	\rangle	>
\xi	ξ	\Theta	Θ	\sim	~
\pi	π	\Lambda	Λ	\leq	≤
\rho	ρ	\Xi	Ξ	\infty	∞
\sigma	σ	\Pi	Π	\clubsuit	♣
\varsigma	ς	\Sigma	Σ	\diamondsuit	♦
\tau	τ	\Upsilon	ϒ	\heartsuit	♥
\equiv	≡	\Phi	Φ	\spadesuit	♠
\Im	ℑ	\Psi	Ψ	\leftrightarrow	↔
\otimes	⊗	\Omega	Ω	\leftarrow	←
\cap	∩	\forall	∀	\uparrow	↑
\supset	⊃	\exists	∃	\rightarrow	→
\int	∫	\ni	∍	\downarrow	↓
\circ	°	\neq	≠	\nabla	∇
\pm	±	\aleph	ℵ	\ldots	…
\geq	≥	\wp	℘	\prime	′
\propto	∝	\oslash	∅	\0	∅
\partial	∂	\supseteq	⊇	\mid	|
\bullet	•	\subset	⊂	\copyright	©
\div	÷	\o	o		

Matlab 还提供了一个 TeX 格式命令的有限子集。上标和下标是分别用^和_声明的。文本字体和字体大小可以用\fontname 和\fontsize 命令进行选择，字体风格可以用\bf、\it、\sl 或者\rm 命令进行声明，分别选择粗体字、斜体字、透明或印刷体，或者普通 Roman 字体。为了输入用来定义 TeX 字符串的特殊字符，在这些字符前面加上一个反斜线符号。受到影响的字符是反斜线（\），左右花括号{}，下划线（_）以及连字符号（^）之间的字符。下边这个例子展示了 TeX 格式命令的用法：

```
>> close % close last Figure window and start over
>> axis([0 1 0 0.5])
```

```
>> text(0.2,0.1,'\itE = M\cdotC^{\rm2}')
>> text(0.2,0.2,'\fontsize{16} \nabla \times H = J + \partialD/\partialt')
>> text(0.2,0.3,'\fontname{courier}\fontsize{16}\bf x_{\alpha} +y^{2\pi}')
>> fsstr='f(t) = A_o + \fontsize{30}_\Sigma\fontsize{10}';
>> text(0.2,0.4,[fsstr '[A_ncos(n\omega_ot) + B_nsin(n\omega_ot)]'])
>> title('Figure 26.22: TeX Formatting Examples')
```



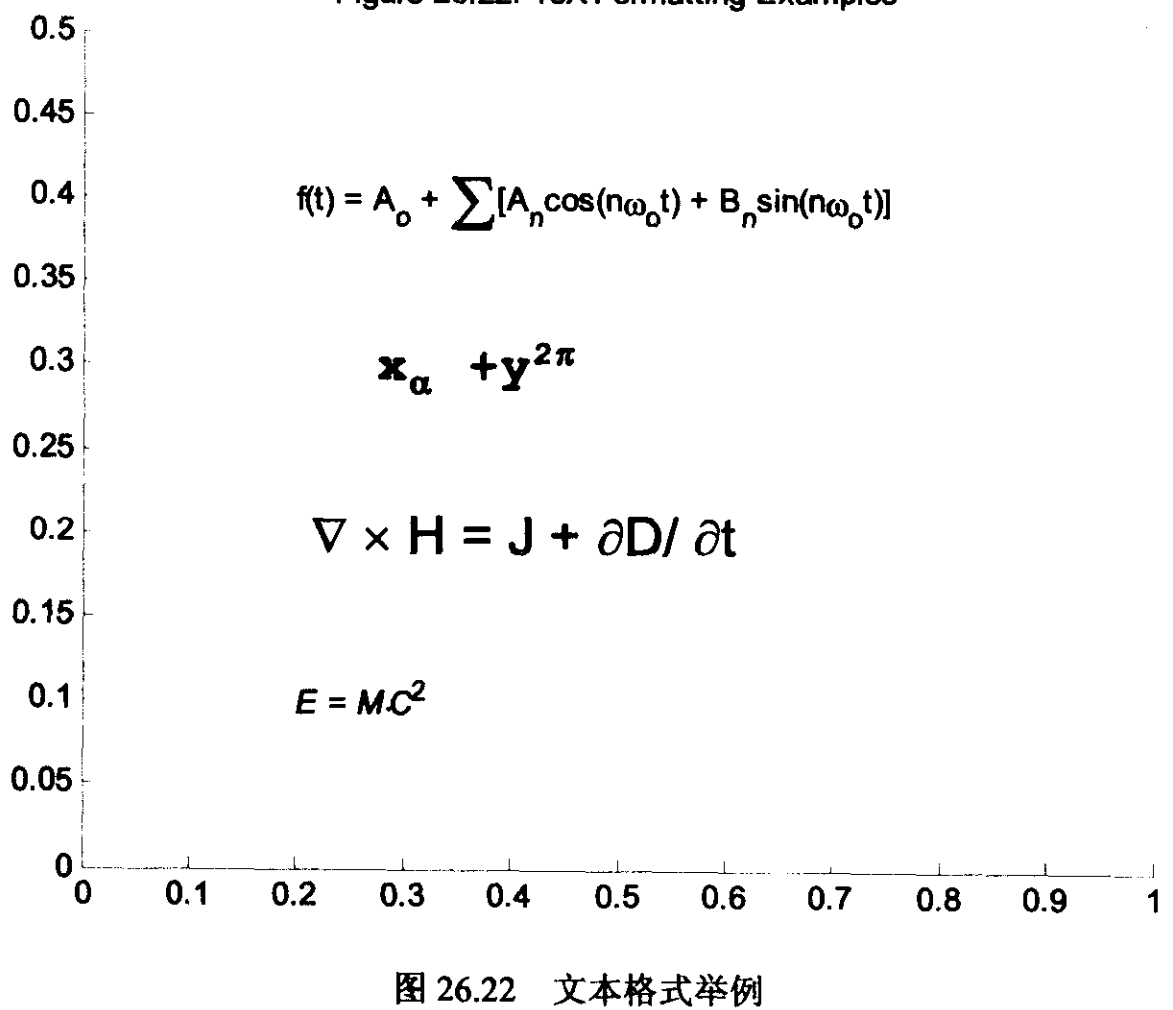


图 26.22 文本格式举例

26.13 小结

下面这个表格列出了 Matlab 中用来进行二维图形绘制的函数。

函数	描述
plot	线性绘图
loglog	对数坐标轴绘图
semilogx	半对数坐标轴（x 轴）绘图
semilogy	半对数坐标轴（y 轴）绘图
polar	极坐标绘图
plotyy	双 y 轴线性绘图
axis	用于控制坐标轴的刻度和外观
xlim	设置 x 轴的坐标范围
ylim	设置 y 轴的坐标范围
zlim	设置 z 轴的坐标范围
daspect	设置和获取数据高宽比，例如，axis equal

（续表）

函数	描述
pbaspect	设置和获取屏幕高宽比，例如，axis square
zoom	图形的放大和缩小
grid	显示和隐藏栅格线
box	显示和隐藏坐标轴边框
hold	保持当前的图形
subplot	用于在同一图形窗口中生成多个坐标轴
figure	用于生成图形窗口
legend	添加图例
title	在图形的顶部添加标题
xlabel	添加 x 轴标注
ylabel	添加 y 轴标注
text	在图形中放置文本
gtext	在鼠标点击处放置文本
ginput	获得鼠标点击处的坐标
area	填充一个图形与横坐标之间的区域
bar	绘制条形图
barh	绘制水平条形图
bar3	绘制三维条形图
bar3h	绘制三维水平条形图
compass	绘制绕行曲线
errorbar	绘制误差线段
ezplot	利用字符串表达式轻松绘制线型图
ezpolar	利用字符串表达式轻松绘制极坐标图
feather	绘制羽状图
fill	绘制实心的二维多边形
fplot	利用给定的函数绘图
hist	绘制直方图
pareto	绘制 Pareto 图
pie	绘制饼状图
pie3	绘制三维饼状图
plotmatrix	绘制矩阵散布图
ribbon	将二维线以线性方式绘制成三维的带状
scatter	绘制分散数据图
stem	绘制离散序列的柄状图
stairs	绘制阶梯图

Chapter 27

三 维 图 形

Matlab 提供了多个函数来显示三维数据，其中有的函数用来绘制三维曲线，有的函数用来绘制三维曲面，而有的函数则用来绘制三维框架。除此以外，用户还可以使用颜色来表示数据的第四维。由于颜色在绘图过程中通常都是图形的一个自然属性，因此，用于表示第四维的颜色在 Matlab 中被称为伪色（pseudocolor）。本章主要讨论三维图形的基本概念，有关伪色的内容将在下一章介绍。

27.1 曲线图

与二维曲线绘制函数 plot 相对应，Matlab 提供了 plot3 函数用于绘制三维曲线。plot3 函数的用法和 plot 函数的用法一样，只是在绘图时需要用户每次提供 3 个数据参数（称为一个数据组）。plot3 函数的常见调用格式为：plot3(x1,y1,z1,S1,x2,y2,z2,S2,...)，其中 xn、yn、zn 为一组向量或矩阵，sn 则是可选的用来声明颜色、标记符号和线型的字符串。plot3 通常用于绘制一个单一变量的三维函数，如下例所示：

```
>> t = linspace(0,10*pi);
>> plot3(sin(t),cos(t),t)
>> xlabel('sin(t)'),ylabel('cos(t)'),zlabel('t')
>> text(0,0,0,'Origin')
>> grid on
>> title('Figure 27.1: Helix')
>> v = axis
v =
    -1      1      -1      1      0      35
```

上一章介绍的二维图形的所有基本特性都可以直接或经简单处理后应用在三维图形中。例如，上例中最后一个命令 axis 返回三维图形的 3 个坐标范围（其中最后两个元素即代表 z 轴的范围）。另外，利用命令 grid 也可以在三维图形中生成三维栅格，利用命令 box 则可以生成一个包围图形的三维边框（与 plot 一样，plot3 的默认设置也为 grid off 和 box off）。用户也可以利用命令 text(x,y,z,'string')将一个字符串'string'放置在由 x、y、z 指定的位置。最后，子图和多图形窗口也可以直接应用到三维图形函数上。

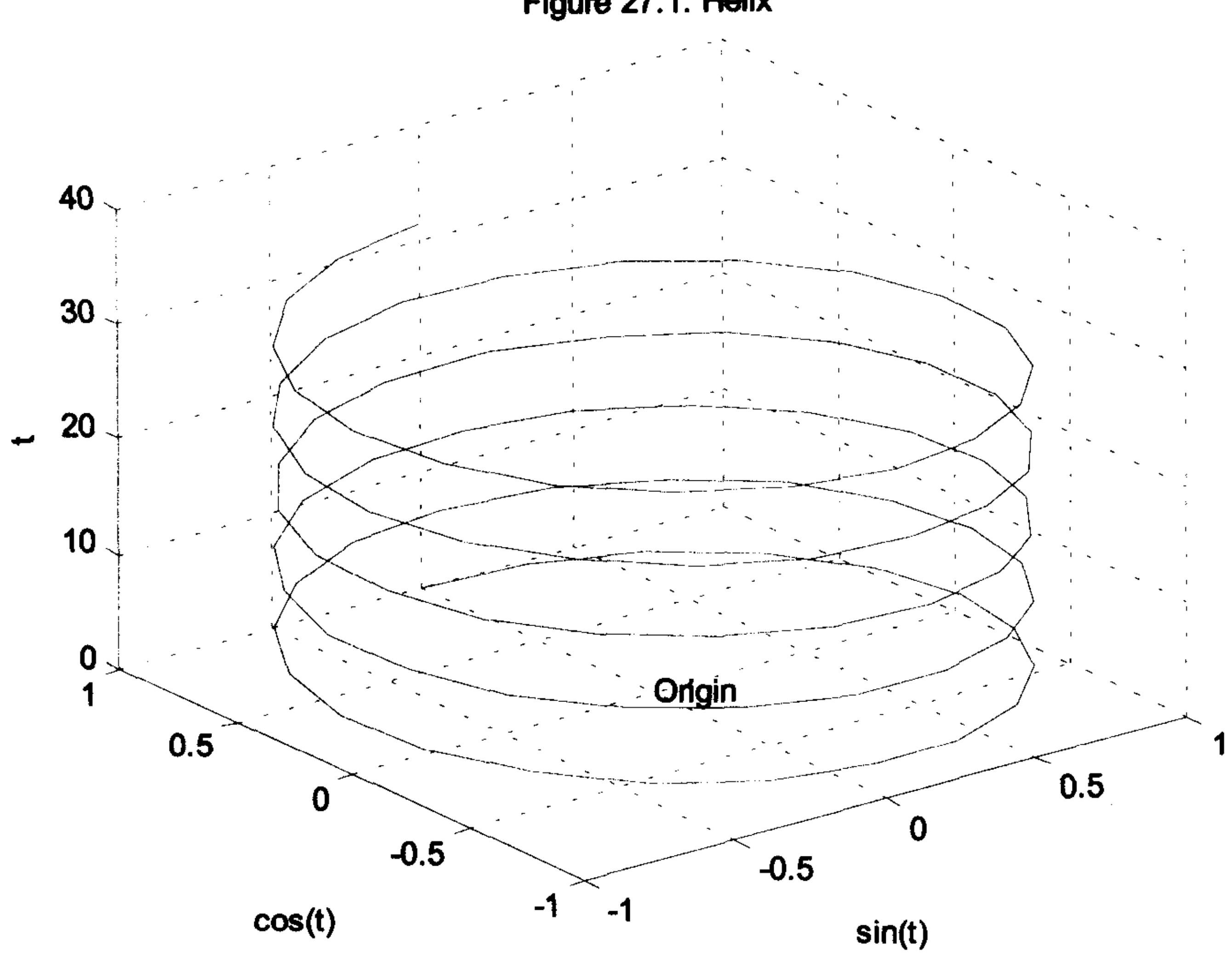


图 27.1 三维螺旋线

在上一章中，通过给 plot 函数提供多个参数对，或者使用 hold 命令，就可以在一个图形窗口中绘制多条曲线。在三维图形窗口中，plot3 以及其他一些三维图形函数也提供了相同的功能。例如，如果给 plot3 函数提供多个数据组，则多个二维图形可以沿着某一维层叠在其他二维图形上。例如，下面的代码利用 plot3 函数绘制了 3 个三维曲线：

```
>> x = linspace(0,3*pi);    % x-axis data
>> z1 = sin(x);             % plot in x-z plane
>> z2 = sin(2*x);
>> z3 = sin(3*x);
>> y1 = zeros(size(x));     % spread out along y-axes
>> y3 = ones(size(x));      % by giving each curve different y-axis values
>> y2 = y3/2;
>> plot3(x,y1,z1,x,y2,z2,x,y3,z3)
>> grid on
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.2: sin(x),sin(2x),sin(3x)')
>> pause(5)
>> plot3(x,z1,y1,x,z2,y2,x,z3,y3)
>> grid on
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.3: sin(x),sin(2x),sin(3x)')
```

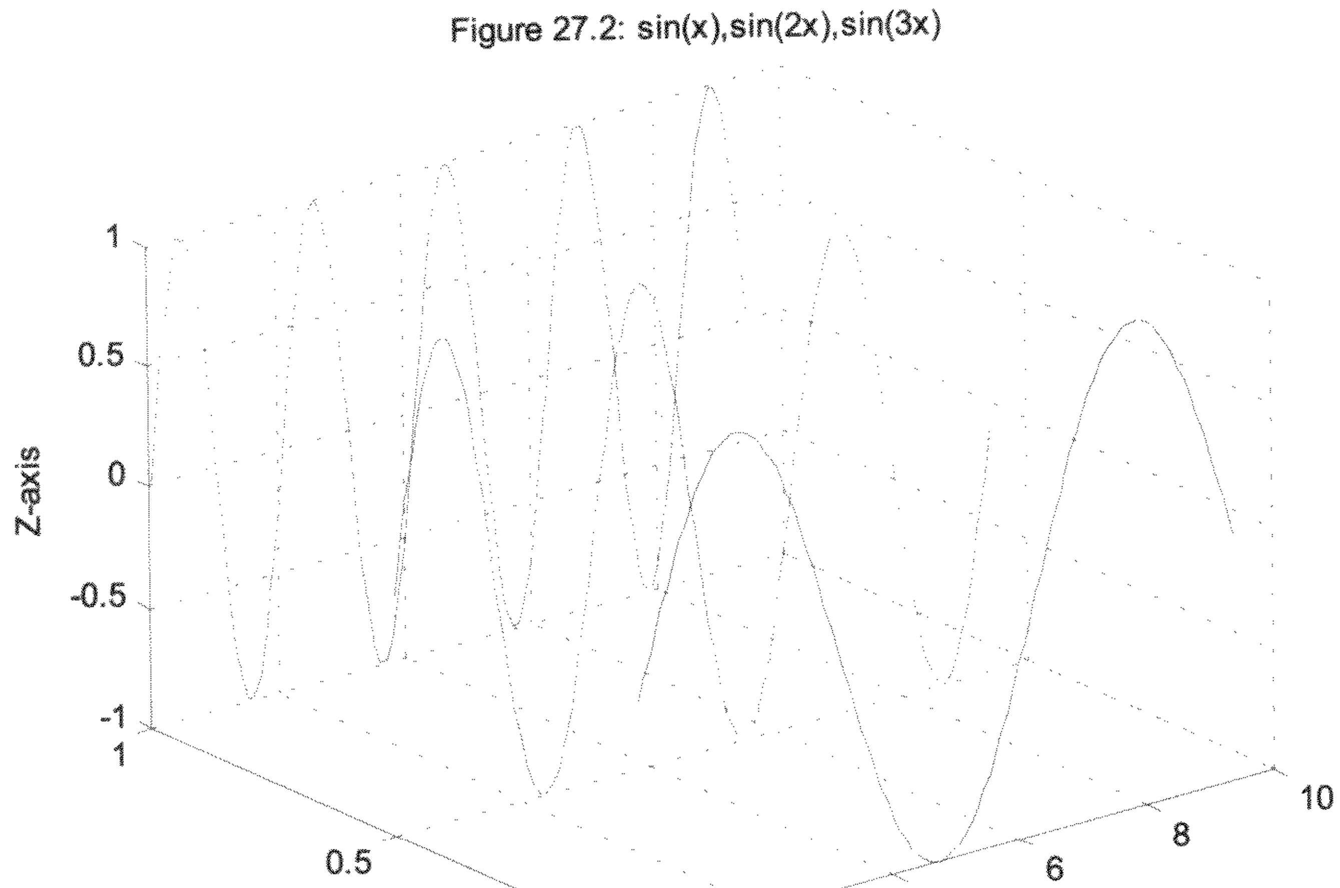


图 27.2　sin(x),sin($2x$),sin($3x$)曲线



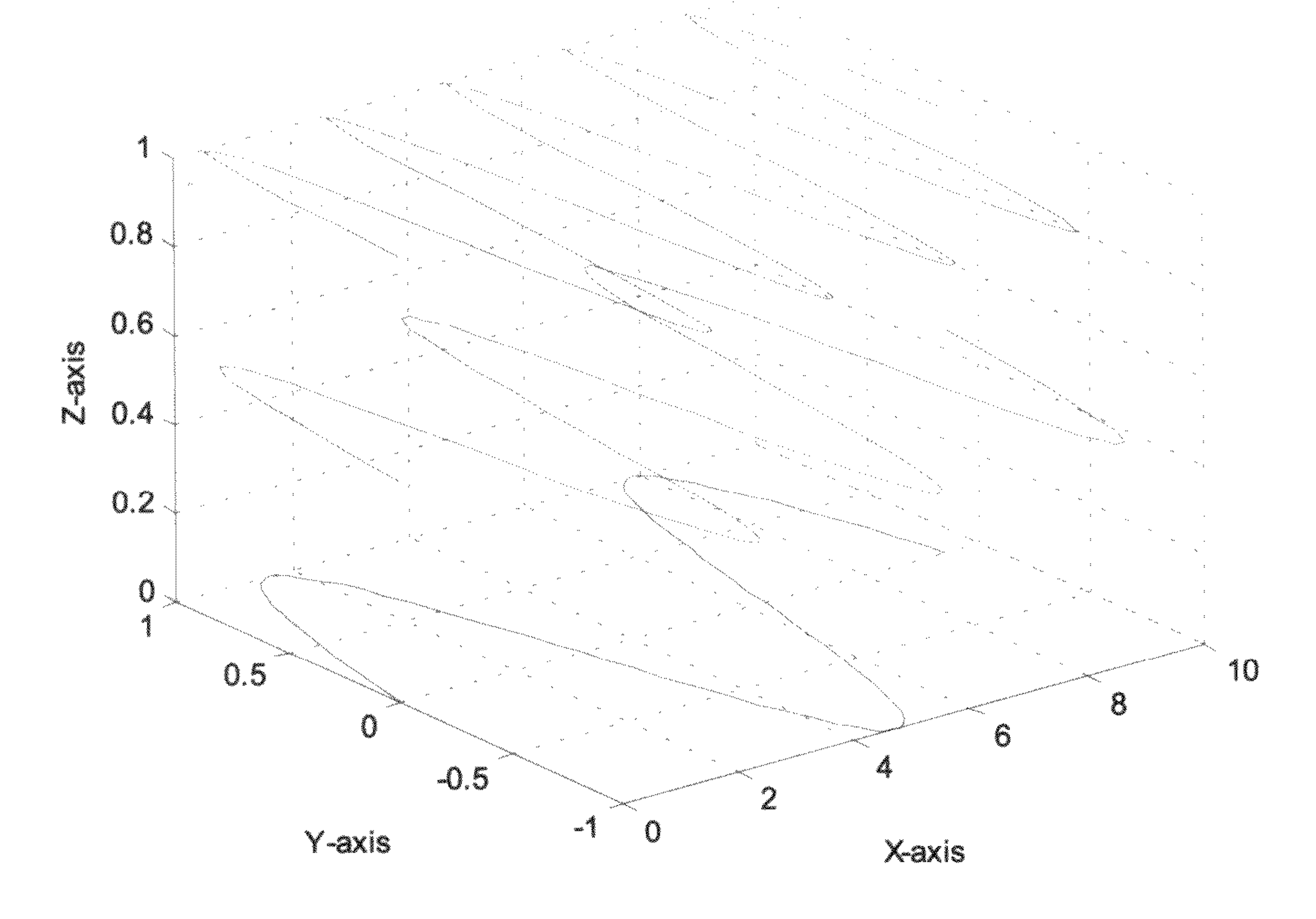


图 27.3　sin(x),sin($2x$),sin($3x$)曲线

27.2 含有两个变量的标量函数

含有两个变量的标量函数的基本格式如下：

$$z = f(x,y)$$

以上函数中，每一对给定的 x 和 y 的值都会产生一个 z 值。

在 Matlab 中，该函数的图形是一个三维曲面。如果将 x 与 y 视为自变量，则 z 就是由这两个自变量生成的一个矩阵。z 与 x、y 之间的关系为：

```
z(i,:) = f(x,y(i))
z(:,j) = f(x(j),y)
```

也就是说，z 的第 i 行和 y 的第 i 个元素相关，而 z 的第 j 列和 x 的第 j 个元素相关。

如果 z=f(x,y)可以用一个简单的表达式表示时，可以直接利用数组运算来计算 z 的值。此时，用户必须首先创建 x 和 y 能够表示的所有的格点，该格点由两个矩阵 X 和 Y 表示，其中 X 包含每个格点的横坐标，Y 包含每个格点的纵坐标。Matlab 提供了函数 meshgrid 来生成这两个矩阵，如下例所示：

```
>> x = -3:3;     % choose x-axis values
>> y = 1:5;      % y-axis values
>> [X,Y] = meshgrid(x,y)
X =
    -3    -2    -1     0     1     2     3
    -3    -2    -1     0     1     2     3
    -3    -2    -1     0     1     2     3
    -3    -2    -1     0     1     2     3
    -3    -2    -1     0     1     2     3
Y =
     1     1     1     1     1     1     1
     2     2     2     2     2     2     2
     3     3     3     3     3     3     3
     4     4     4     4     4     4     4
     5     5     5     5     5     5     5
```

从上面的结果可以看到，meshgrid 函数将行向量 x 复制 5 次（y 的列数），将列向量 y 复制 7 次（x 的列数）。

> 如果用户不容易记住 meshgrid 到底将 x 或 y 变量复制多少次，可以想象一下在 x-y 平面上，每个给定的 x 中的元素值都会有 length(y) 个 y 的元素值与之对应，因此，要形成一个完整的栅格结构，x 必须被复制 length(y) 次；而每个给定的 y 中的元素值都会有 length(x) 个 x 的元素值与之对应，y 必须被复制 length(x) 次。

假设 $z=f(x,y)=(x+y)^2$，那么我们就可以使用上例得出的 X 和 Y 来计算 Z 了：

```
>> Z = (X+Y).^2
Z =
     4     1     0     1     4     9    16
```

```
     1     0     1     4     9    16    25
     0     1     4     9    16    25    36
     1     4     9    16    25    36    49
     4     9    16    25    36    49    64
```

如果一个函数无法写成如上的简单表达式，那么用户就必须使用 For 循环或 While 循环来计算 Z 的元素值。有时候，也可以整行或整列计算 Z 的元素值。例如，下面给出的命令框架可以整行计算 Z 的值（用户需要在相应的地方输入数据和代表 Z 与 XY 函数关系的代码）：

```
x= ???  % statement defining vector of x-axis values
y= ???  % statement defining vector of y-axis values

nx = length(x);    % length of x is no. of rows in Z
ny = length(y);    % length of y is no. of columns in Z
Z = zeros(nx,ny);  % initialize Z matrix for speed

for r=1:nx
   (preliminary commands)
   Z(r,:)= {a function of y and x(r) defining r-th row of Z}
end
```

下面给出的命令框架可以整列计算 Z 的值（用户也需要在相应的地方输入数据和代表 Z 与 XY 函数关系的代码）：

```
x= ???  % statement defining vector of x-axis values
y= ???  % statement defining vector of y-axis values

nx = length(x);     % length of x is no. of rows in Z
ny = length(y);     % length of y is no. of columns in Z
Z = zeros(nx,ny);   % initialize Z matrix for speed

for c =1:ny
   (preliminary commands)
   Z(:,c)= {a function of y(c) and x defining c-th column of Z}
end
```

下面给出的命令框架可以逐元素计算 Z 的值，其中用到了 For 循环嵌套（用户也需要在相应的地方输入数据和代表 Z 与 XY 函数关系的代码）：

```
x= ???  % statement defining vector of x-axis values
y= ??? % statement defining vector of y-axis values
nx = length(x);         % length of x is no. of rows in Z
ny = length(y);         % length of y is no. of columns in Z
Z = zeros(nx,ny);       % initialize Z matrix for speed
for r=1:nx
    for c=1:ny
        (preliminary commands)
```

```
        Z(r,c)={a function of y(c) and x(r) defining (r,c)-th element}
    end
end
```

27.3 网格图

Matlab 用位于 x-y 平面上方的 z 的坐标来定义一个网格面。它通过将相邻的点用直线连接来构成了一个网格图。网格图看上去有点像一个渔网，其网格节点是 z 中的数据点。用于绘制网格图的函数是 mesh，下面给出了一个绘制网格图的例子：

```
>> [X,Y,Z] = peaks(30);
>> mesh(X,Y,Z)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.4: Mesh Plot of Peaks')
```



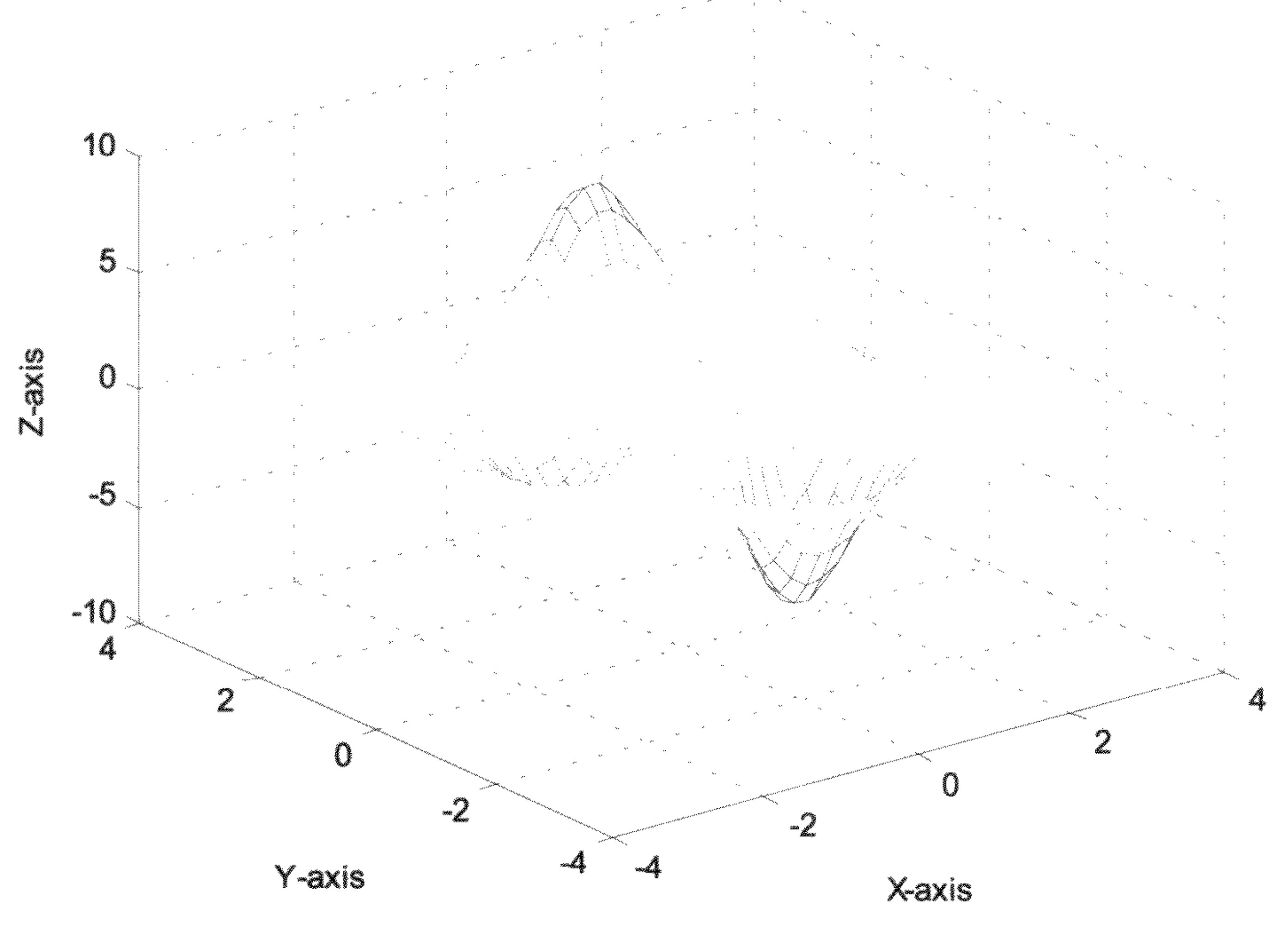


图 27.4 网格图

如果用户在显示器上观察图 27.4，可以发现随着网格图的高度不同，线条的颜色也不同。如果想改变线条的颜色，用户可在 mesh 函数的可选参数中设置该颜色值。有关颜色设置的内容将在下一章中讨论。另外还需要注意的是，图 27.4 中绘制的图形是显示格栅的。在 Matlab 中，除了 plot3 函数之外，其他大多数函数在绘制三维图形以及一些其他图形时都会将栅格属性默认设置为 grid on。

前面对 mesh 函数的调用是最简单的一种调用方式，除了上边的调用方式外，mesh 函数还有其他一些调用方式。例如，具有单独输入参数的 mesh(Z)方式可以绘制出矩阵 Z 相对于其行索引和列索引的三维图形；另外，我们也可以直接将 x 和 y 轴向量传递给 mesh 函数，而不需要使用 meshgrid 函数进行扩展，如，mesh(x,y,Z)。

在图 27.4 中，我们发现网格图在默认情况下是不透明的，这样只能显示前面的网格线，后面的网格线则被遮挡。如果要使网格图透明显示，可以使用 hidden on 命令，当然，使用 hidden off 命令则使网格图不透明显示。下面的代码演示了网格图的透明属性：

```
>> [X,Y,Z]=sphere(12);
>> subplot(1,2,1)
>> mesh(X,Y,Z),title('Figure 27.5a: Opaque')
>> hidden on
>> axis square off
>> subplot(1,2,2)
>> mesh(X,Y,Z),title('Figure 27.5b: Transparent')
>> hidden off
>> axis square off
```

Figure 27.5a: Opaque

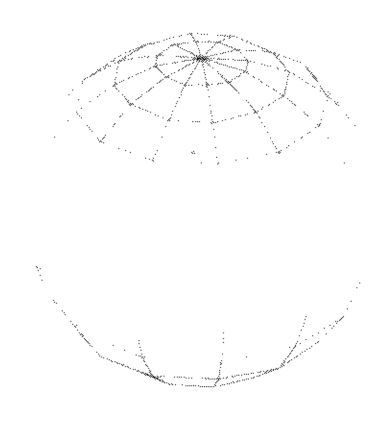


Figure 27.5b: Transparent

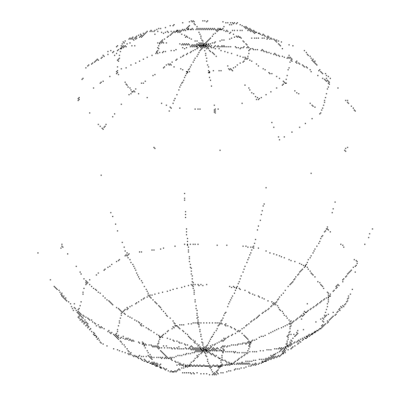


图 27.5 透明属性

在上面的图中，图 27.5a 中的球体是不透明的（后边的线条被隐藏了），而图 27.5b 中的球体是透明的（后边的线条没有被隐藏）。

另外，Matlab 还提供了两个变体函数，一个是 meshc，它用来绘制用等值线描述的网格图；另一个是 meshz，它用来绘制一个包含了 0 平面的网格图。下面的例子给出了这两个函数的应用：

```
>> [X,Y,Z] = peaks(30);
>> meshc(X,Y,Z)     % mesh plot with underlying contour plot
>> title('Figure 27.6: Mesh Plot with Contours')
>> pause(5)
>> meshz(X,Y,Z)     % mesh plot with zero plane
>> title('Figure 27.7: Mesh Plot with Zero Plane')
```



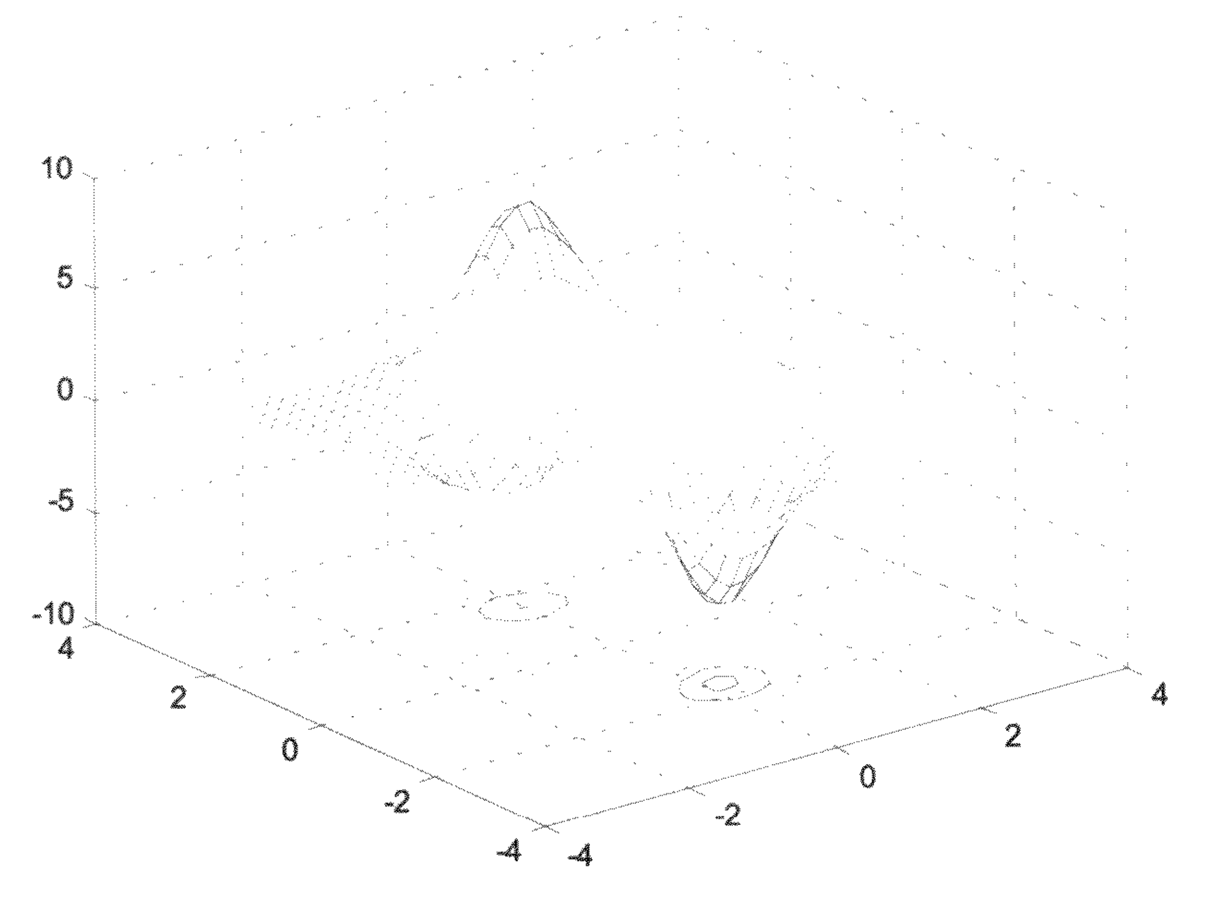


图 27.6 带等高线的网格图

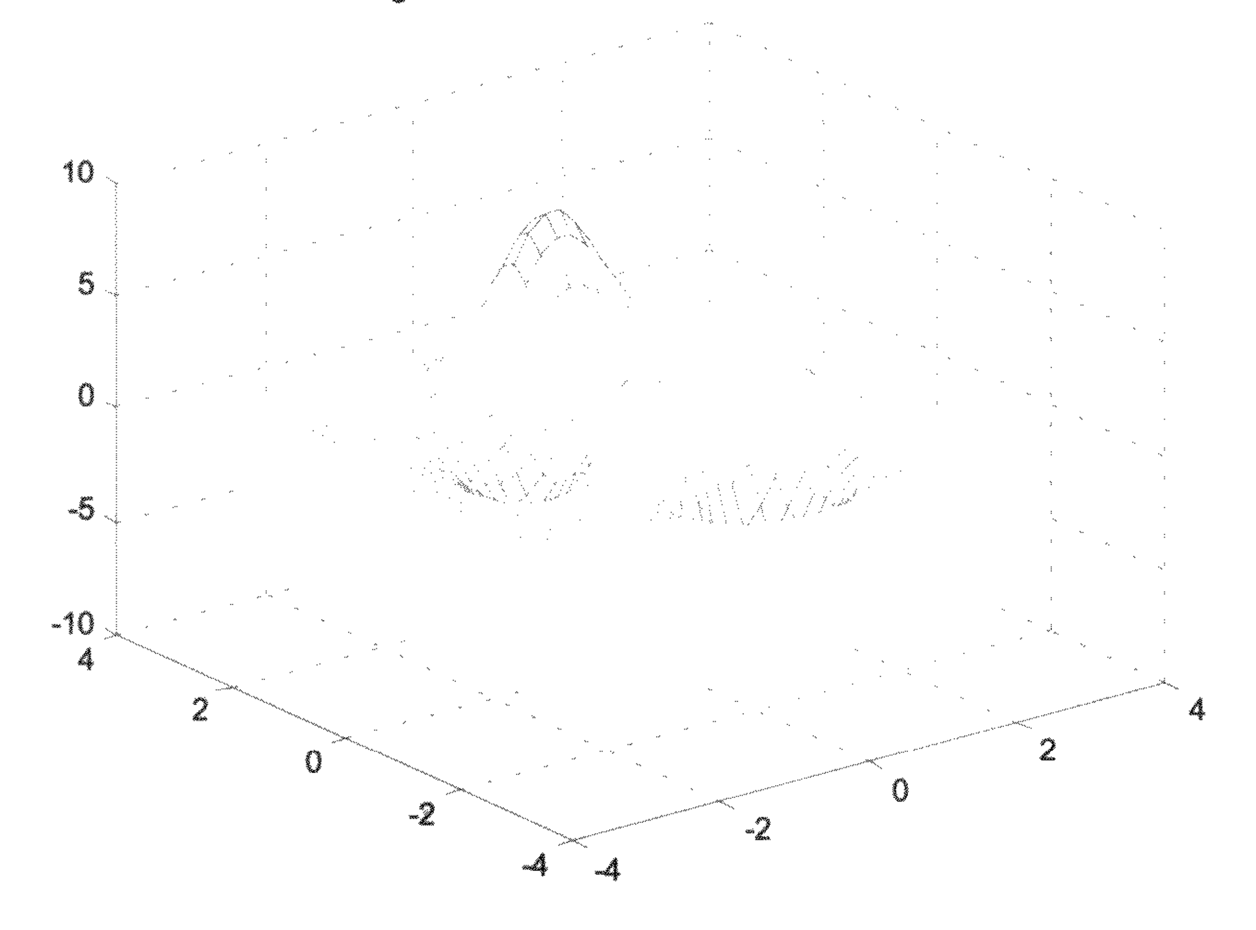


图 27.7 带零平面的网格图

函数 waterfall 与 mesh 的用法基本相同，只是其只绘制 x 轴方向的网格线（因其绘制的图形酷似瀑布，因此函数名为 waterfall），如下例所示：

```
>> waterfall(X,Y,Z)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.8: Waterfall Plot')
```

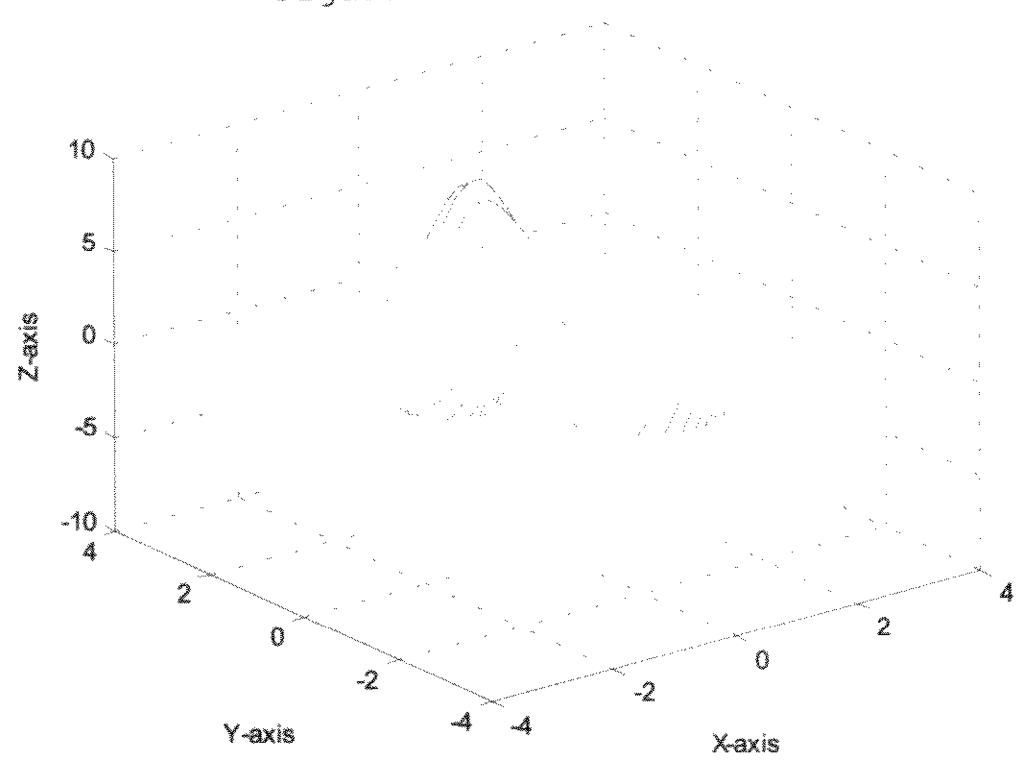


图 27.8 瀑布图

27.4 表面图

表面图用来表示三维图形的表面，它与网格图类似，只是其网格都被填充了颜色。在 Matlab 中，用户可以使用 surf 函数绘制表面图，如下例所示：

```
>> [X,Y,Z] = peaks(30);
>> surf(X,Y,Z)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.9: Surface Plot of Peaks')
```

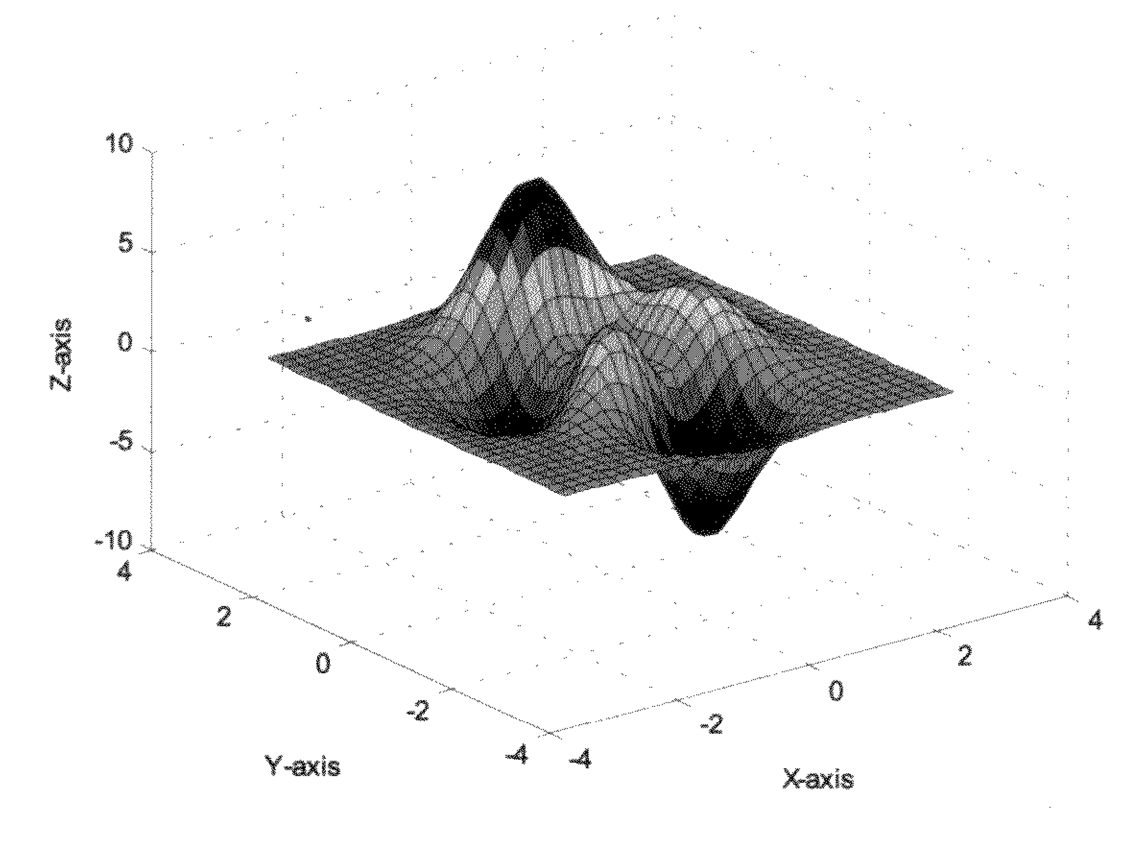


图 27.9 表面图

读者可以发现，表面图与网格图在颜色绘制上具有某种形式的对偶性。比如，表面图的线条是单一的颜色（本例为黑色），而网格被填充了不同的颜色；但在网格图中，网格被填充了单一的颜色（如图 27.6 中为白色），而线条被绘制了不同的颜色。表面图网格的颜色也根据其高度不同发生变化，这与网格图中的线条颜色变化特性一样。另外，在默认情况下，表面图中含有栅格线，即栅格属性为 grid on。

在网格图中，用户要考虑如何将被隐藏的线条显示，而在表面图中，就要考虑如何使表面具有阴影（Shading）效果。在 surf 图形中，阴影就像大厦外面的彩色玻璃窗那样是分成许多方块的，其中黑色的线条就是各方块之间的接缝。除了分块阴影外，Matlab 还提供了平面阴影以及插值阴影，它们都可以通过函数 shading 实现。例如，下面的代码分别实现了 surf 表面的平面阴影和插值阴影：

```
>> [X,Y,Z] = peaks(30);
>> surf(X,Y,Z)  % same plot as above
>> shading flat
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.10: Surface Plot with Flat Shading')
>> pause(5)
>> shading interp
>> title('Figure 27.11: Surface Plot with Interpolated Shading')
```

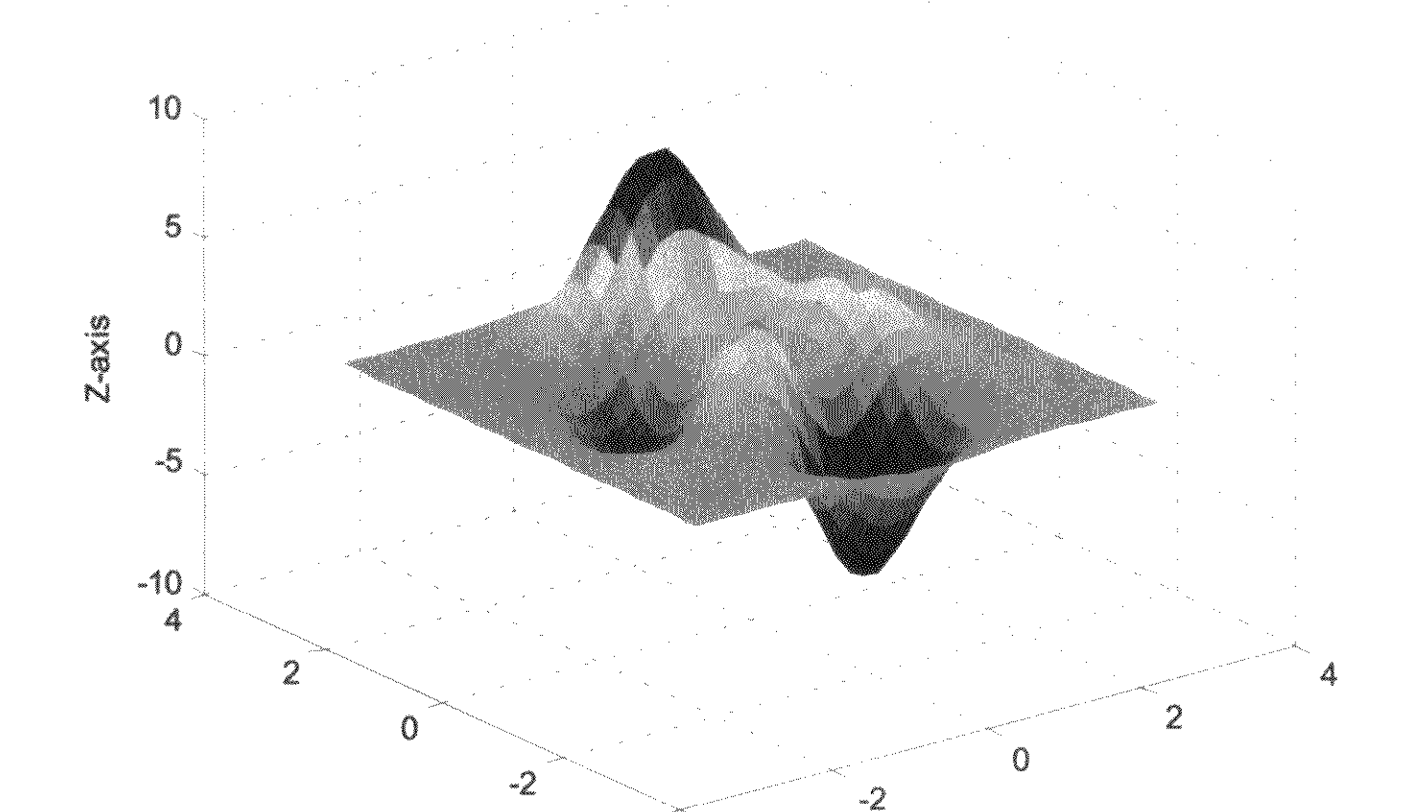


图 27.10 表面图的平面阴影效果

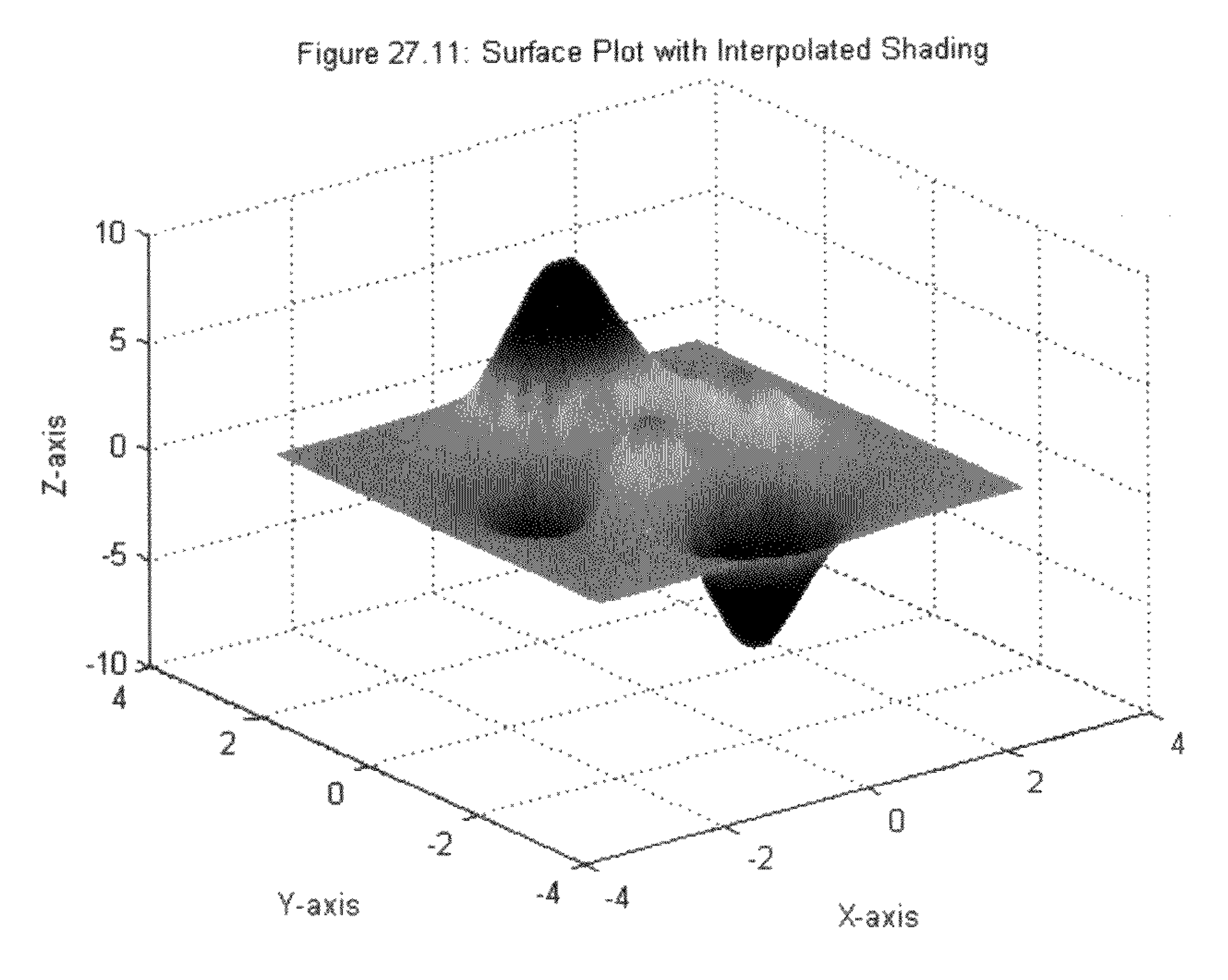


图 27.11 表面图的插值阴影效果

在图 27.10 中的平面阴影中，表面图中的黑色线条都被去掉了，但每个网格都保持了它原来的颜色；而图 27.11 中的插值阴影中，网格之间因为插值阴影颜色变得比较连续。很明显，插值阴影比分块阴影和平面阴影需要更多的计算量。阴影对于一个图形的表现具有重要的视觉影响，因此，它也被应用到了网格图中。但由于网格图只有线条有颜色，因此阴影视觉效果相对来说要小一些。另外，阴影还会对 pcolor 和 fill 图形产生影响。

在有的计算机系统中，插值阴影产生的图形在打印时会产生特别长的延迟，有时甚至导致打印错误。这些问题通常与 PostScript 数据文件无关，而是因为为了生成连续变化的阴影在打印机中所需要进行的大量运算。解决这个问题的最简单的办法是在需要打印输出时，尽量采用平面阴影图。

表面图不像网格图，可以通过设置透明属性观察被遮挡的部分。在表面图中，要想看到被遮挡的部分，必须将前面的表面部分删除。Matlab 没有提供专门用于删除某一块表面图的函数，但却提供了一个在表面图上“打洞”的方法，即将“打洞”位置的数据值设定为 NaN。在 Matlab 绘图时，会忽略所有的 NaN 数据点，这样就在 NaN 出现的位置留出一个“空洞”。例如，下面的例子在 surf 图上打了一个“洞”：

```
>> [X,Y,Z] = peaks(30);
>> x = X(1,:);              % vector of x-axis
>> y = Y(:,1);              % vector of y-axis
>> i = find(y>.8 & y<1.2); % find y-axis indices of hole
>> j = find(x>-.6 & x<.5); % find x-axis indices of hole
>> Z(i,j) = nan;            % set values at hole indices to NaNs
>> surf(X,Y,Z)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.12: Surface Plot with a Hole');
```



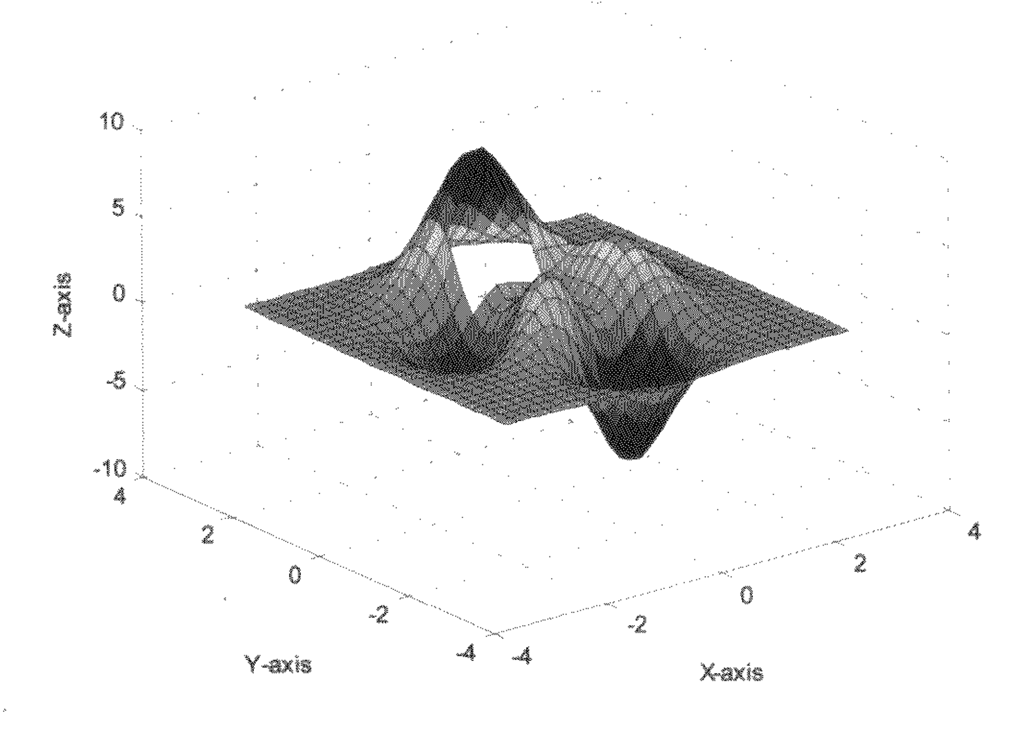


图 27.12 在表面图上“打洞”

surf 函数也有两个变体函数：一个是 surfc，它在绘制表面图时还绘制了底层等高线图；另一个是 surfl，它在绘图时考虑到了光照效果。下面的代码展示了这两个函数的应用：

```
>> [X,Y,Z]= peaks(30);
>> surfc(X,Y,Z) % surf plot with contour plot
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.13: Surface Plot with Contours')
>> pause(5)
>> surfl(X,Y,Z)     % surf plot with lighting
>> shading interp   % surfl plots look best with interp shading
>> colormap pink    % they also look better with shades of a single color
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.14: Surface Plot with Lighting')
```

Figure 27.13: Surface Plot with Contours

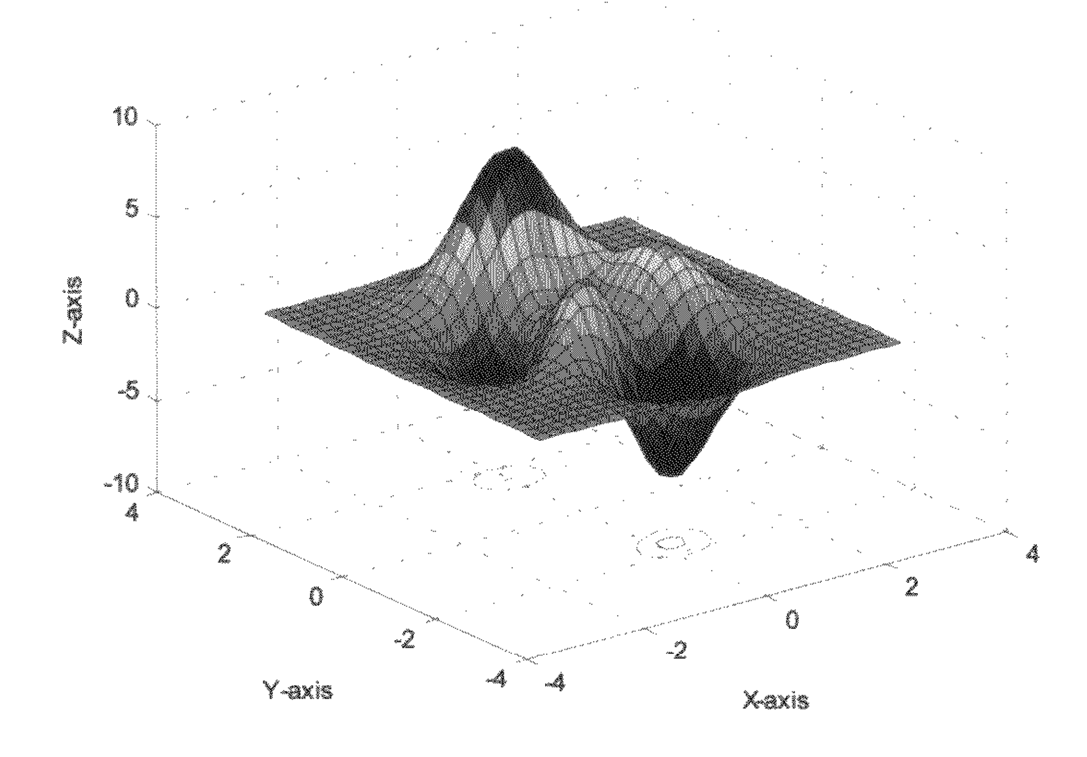


图 27.13 带等高线的表面图

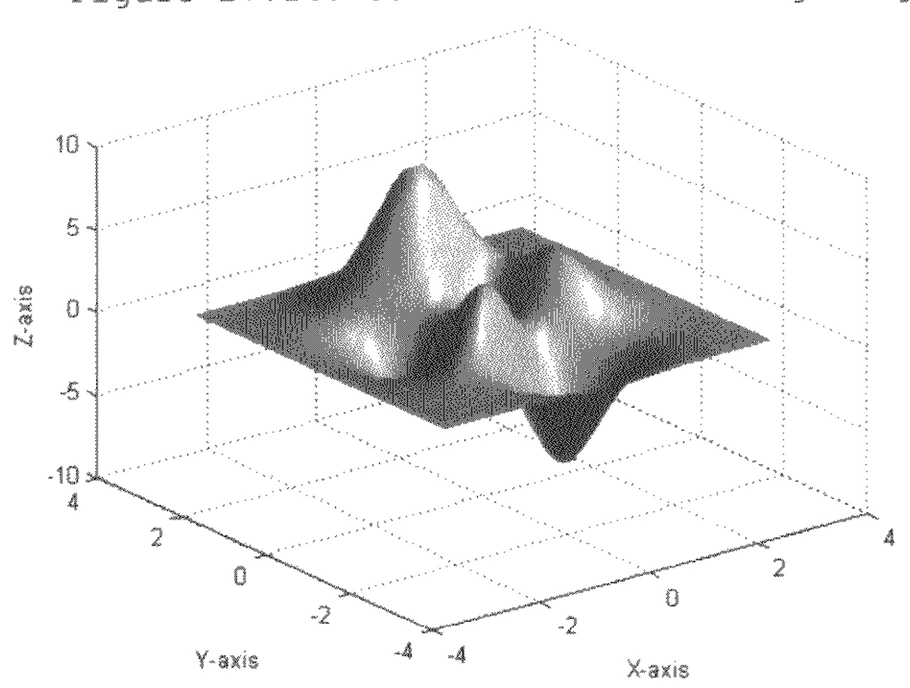


图 27.14 在表面图中使用光照效果

在图 27.14 中，函数 surfl 利用了一系列假设（如光照的方向、强度、阴影颜色等）来绘制表面的光照效果，它并没有用到下一章将要讨论的光照（light）对象。实际上，surfl 函数仅仅是将表面的颜色进行了修改以便得到光照的效果。有关光照对象及其属性将在下一章进行详细阐述。另外，在前面的命令中，colormap 是一个 Matlab 颜色处理函数，用来将不同的颜色集应用到一个图形上，这个函数也将在下一章中进行讨论。

函数 surfnorm(X,Y,Z)用于计算由数据 X、Y 和 Z 定义的表面的法线。该函数将绘制出表面，同时在数据点位置绘制出表面的法线向量，如下面的代码所示：

```
>> [X,Y,Z] = peaks(15);
>> surfnorm(X,Y,Z)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.15: Surface Plot with Normals')
```

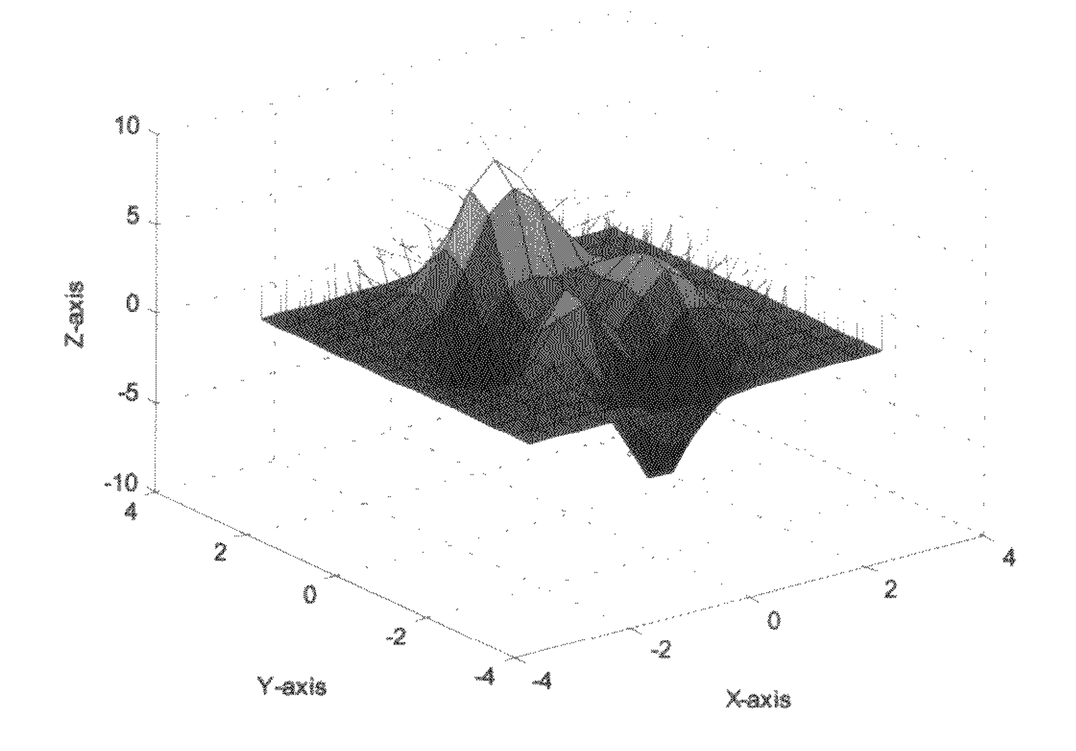


图 27.15 表面图的法线

注意，图 27.15 中的表面法线是没有经过归一化的，并且在每个定点都存在。如果用户只希望返回法线的数据，不想绘制图形，可以采用下面的调用方法：[Nx,Ny,Nz]=surfnorm(X,Y,Z)。

27.5 不规则数据的网格图和表面图

有时候，我们需要对不规则或非均匀分布的数据进行可视化显示，这时就要用到函数 trimesh、trisurf 和 voronoi。下面是这些函数的一个简单应用：

```
>> x = rand(1,50);
>> y = rand(1,50);
>> z = peaks(x,y*pi);
>> t = delaunay(x,y);
>> trimesh(t,x,y,z)
>> hidden off
>> title('Figure 27.16: Triangular Mesh Plot')
>> pause(5)
>> trisurf(t,x,y,z)
>> title('Figure 27.17: Triangular Surface Plot')
>> pause(5)
>> voronoi(x,y)
>> title('Figure 27.18: Voronoi Plot')
```

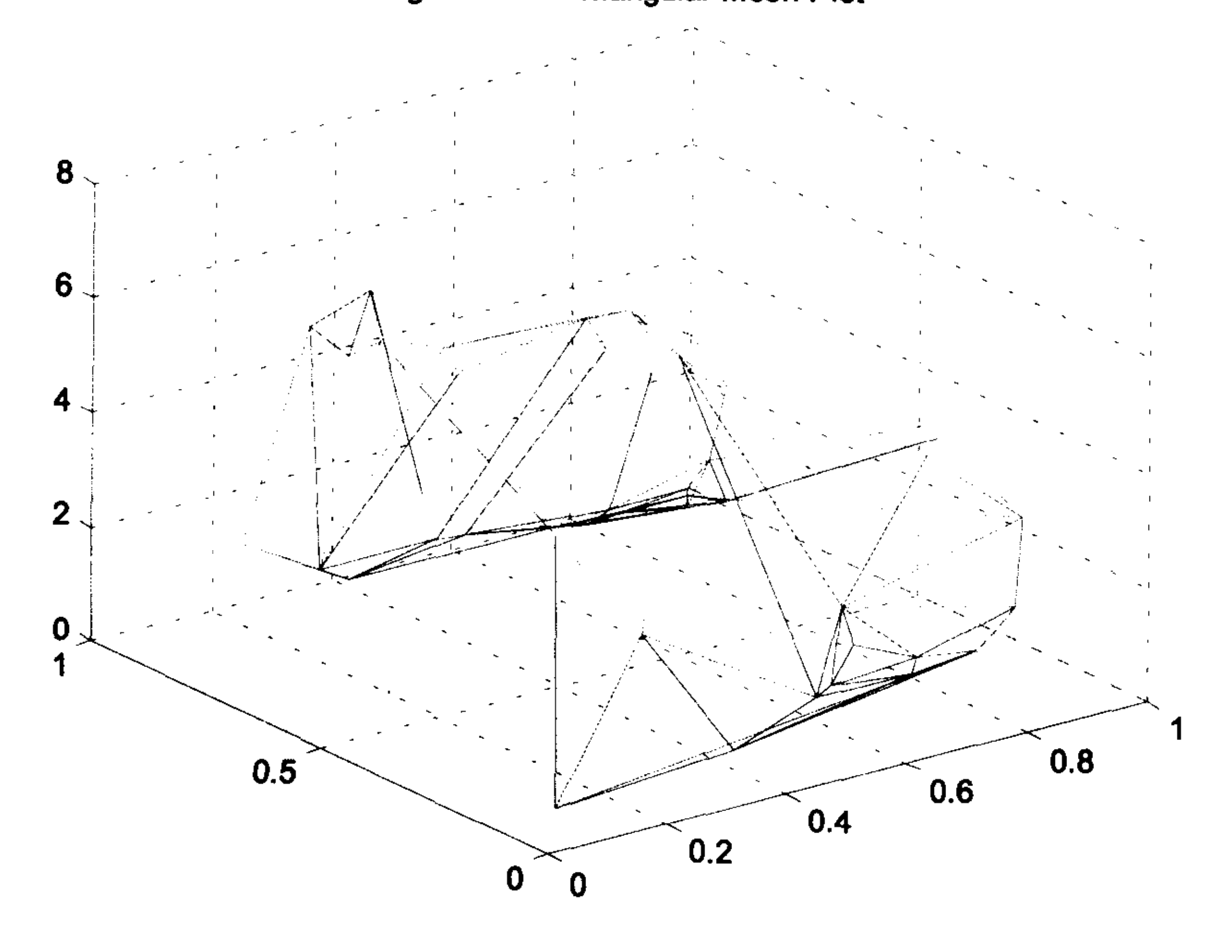


图 27.16 三角网格图



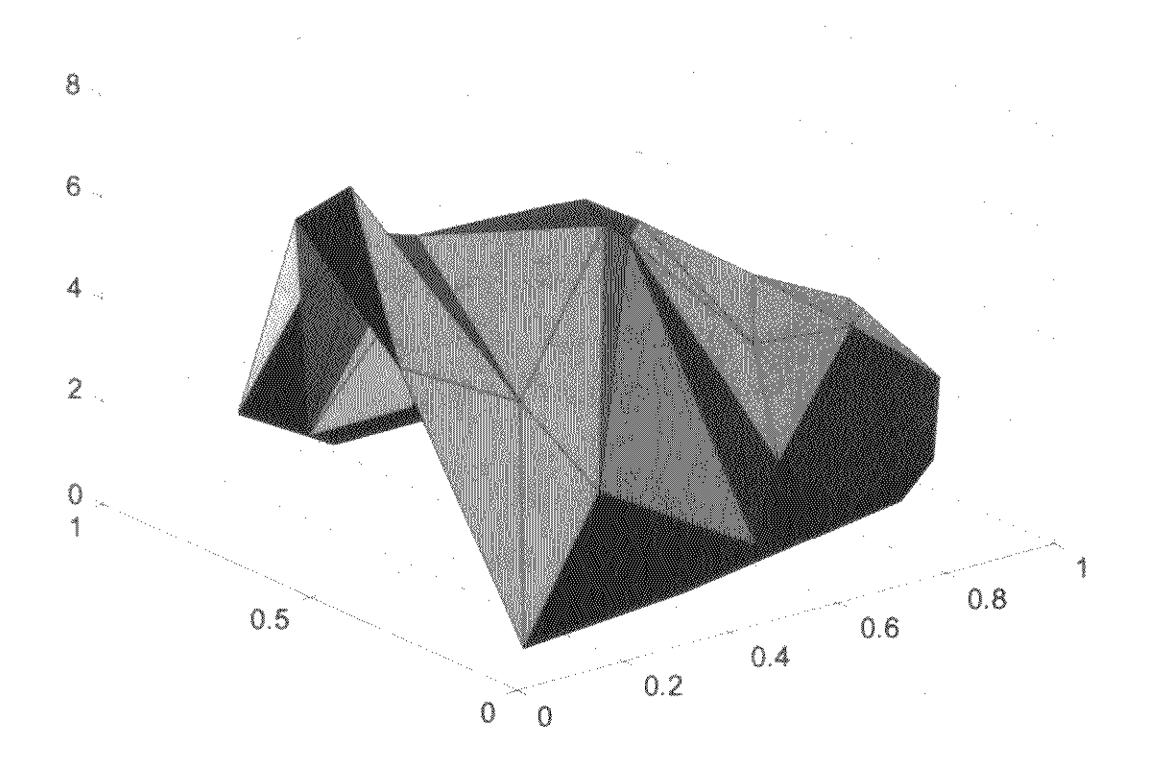


图 27.17　三角表面图

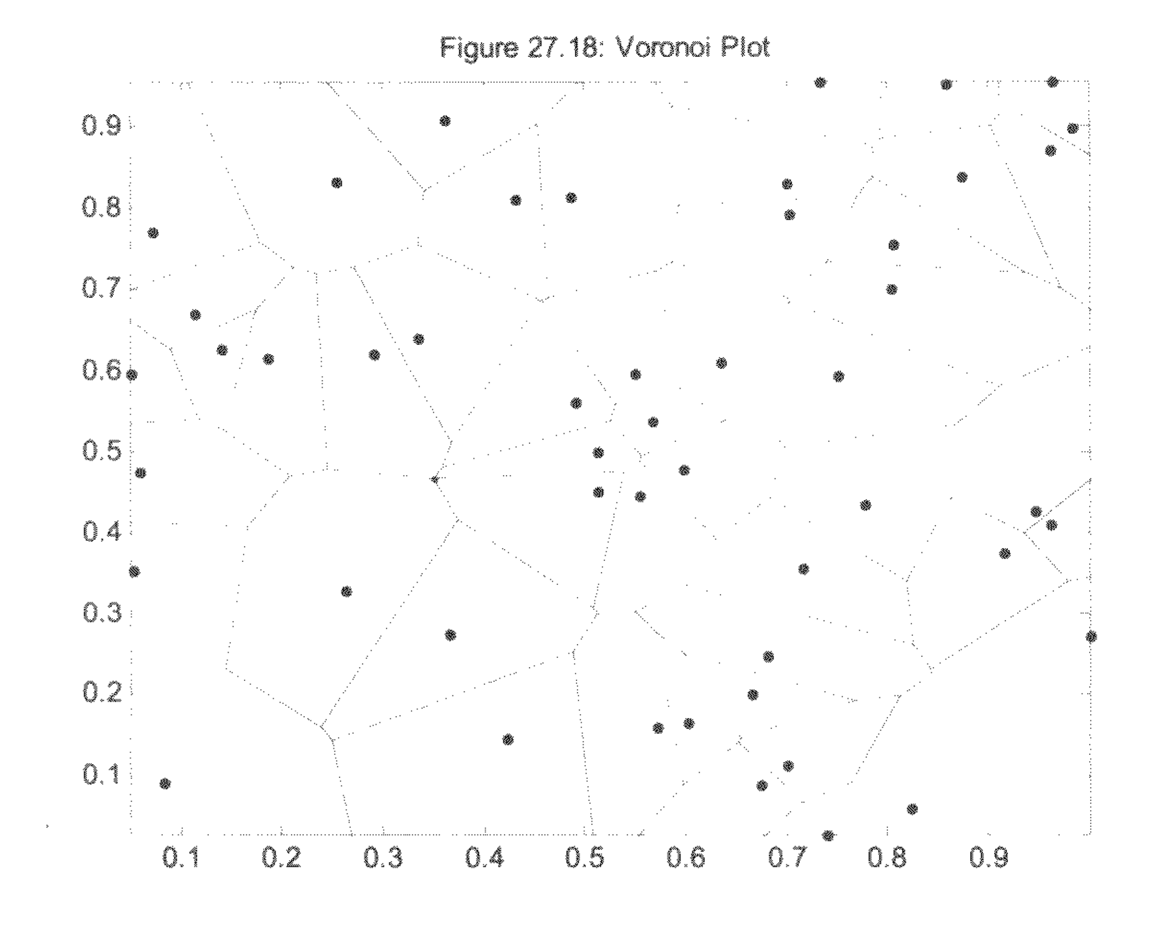


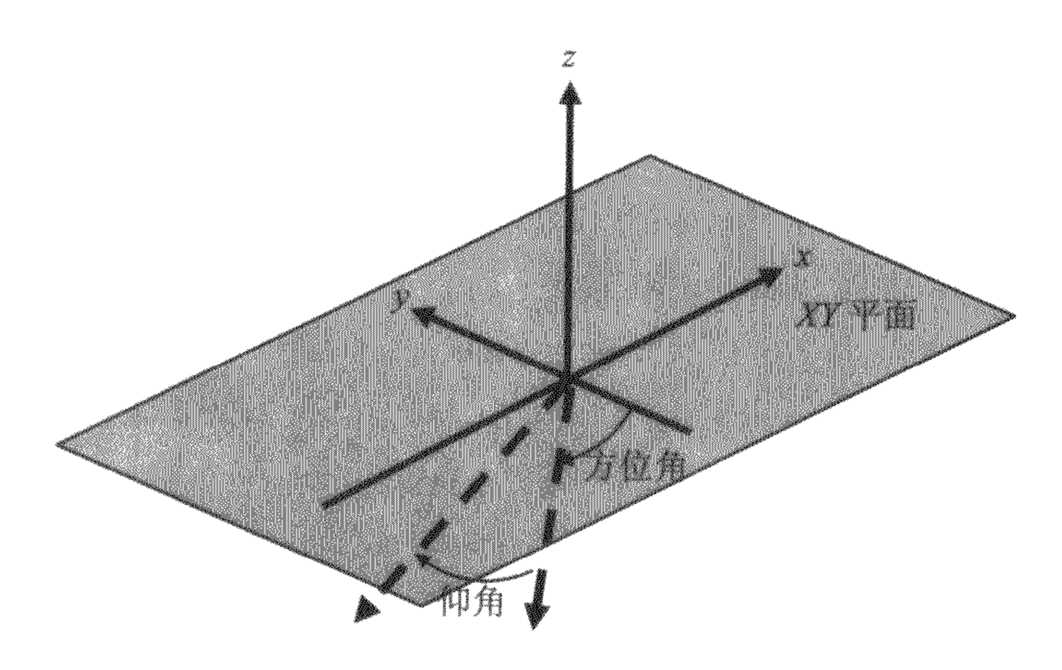
图 27.18　Voronoi 分割图

关于 Delaunay 三角形和 Voronoi 图形的详细用法请参看本书第 19 章。

27.6　改变视角

默认情况下，三维图形的视角是以 30 度的角度向下俯看 z=0 平面，并以 37.5 度的角

度向上仰视 x=0 平面。其中，与 z=0 平面相关的俯看角度被称作仰角，而与 x=0 平面有关的仰视角度被称作方位角。因此，在 Matlab 中，默认的三维视角就是仰角为 30 度、方位角为-37.5 度。同时，仰角和方位角也可用于二维视角，在默认情况下，二维视角为仰角 90 度、方位角 0 度。方位角和仰角的概念可以用下面的图形形象地描述。下面的图形描述了仰角和方位角的具体定义。



在 Matlab 中，用于改变二维或三维图形的视角的函数为 view 函数。view 函数的调用方式为：view(az,el)或 view([az,el])，它将视角改变为指定的方位角 az 和仰角 el。请看下边的例子：

```
>> x = -7.5:.5:7.5; y = x;          % create a data set
>> [X,Y] = meshgrid(x,y);
>> R = sqrt(X.^2+Y.^2)+eps;
>> Z = sin(R)./R;
>> subplot(2,2,1)
>> surf(X,Y,Z)
>> view(-37.5,30)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.19a: Default Az = -37.5,El = 30')

>> subplot(2,2,2)
>> surf(X,Y,Z)
>> view(-37.5+90,30)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.19b: Az Rotated to 52.5')

>> subplot(2,2,3)
>> surf(X,Y,Z)
>> view(-37.5,60)
>> xlabel('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.19c: El Increased to 60')

>> subplot(2,2,4)
>> surf(X,Y,Z)
>> view(0,90)
>> xlabel('X-axis'),ylabel('Y-axis')
>> title('Figure 27.19d: Az = 0,El = 90')
```

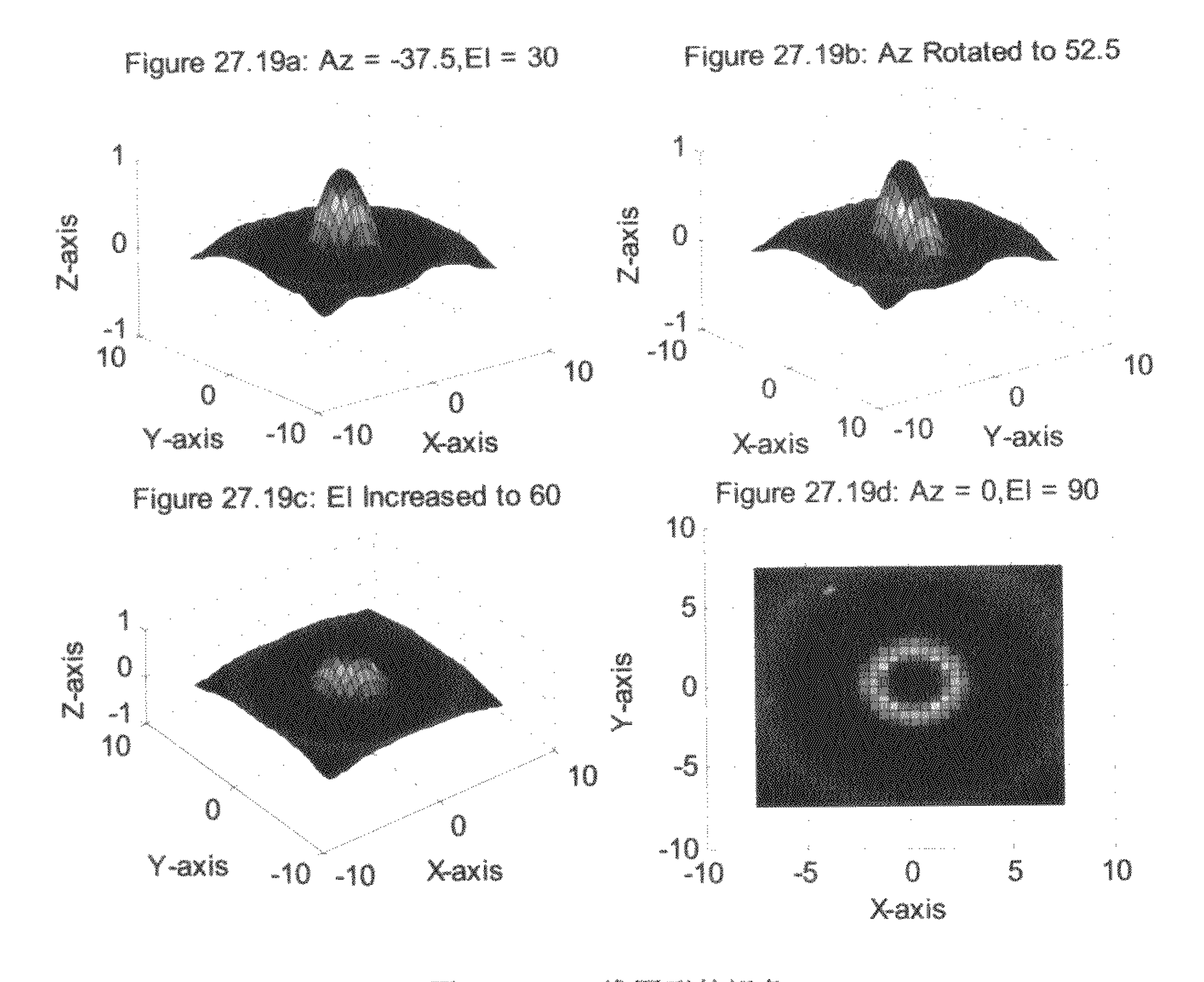


图 27.19　三维图形的视角

除了上边的调用形式外，view 还具有其他的一些特性和使用方法，我们可以通过 view 的帮助文档对这些特性和方法进行了解：

```
>> help view
VIEW   3-D graph viewpoint specification.
   VIEW(AZ,EL) and VIEW([AZ,EL]) set the angle of the view from which an
   observer sees the current 3-D plot.  AZ is the azimuth or horizontal
   rotation and EL is the vertical elevation (both in degrees). Azimuth
   revolves about the z-axis, with positive values indicating counter-
   clockwise rotation of the viewpoint. Positive values of elevation
   correspond to moving above the object; negative values move below.
   VIEW([X Y Z]) sets the view angle in Cartesian coordinates. The
   magnitude of vector X,Y,Z is ignored.

   Here are some examples:

   AZ = -37.5, EL = 30 is the default 3-D view.
   AZ = 0, EL = 90 is directly overhead and the default 2-D view.
   AZ = EL = 0 looks directly up the first column of the matrix.
   AZ = 180 is behind the matrix.

   VIEW(2) sets the default 2-D view, AZ = 0, EL = 90.
   VIEW(3) sets the default 3-D view, AZ = -37.5, EL = 30.

   [AZ,EL] = VIEW returns the current azimuth and elevation.

   VIEW(T) accepts a 4-by-4 transformation matrix, such as
   the perspective transformations generated by VIEWMTX.
```

```
T = VIEW returns the current general 4-by-4 transformation matrix.

See also VIEWMTX, the AXES properties View, Xform.
```

除了 view 函数之外，Matlab 还提供了命令 rotate3d 使用户能够使用鼠标交互式地设置视角。rotate3d on 命令用于打开鼠标设置视角功能，rotate3d off 命令用于关闭该功能，而不带参数的 rotate3d 命令则实现两个状态之间的切换。另外，鼠标设置视角功能还可以从图形窗口中的 Tools 菜单栏中获得，也可以通过选择图形工具栏中的按钮来获得。实际上，无论是菜单项还是工具栏按钮在其内部都调用了 rotate3d 命令来实现用户需要的操作。

27.7 控制摄像机

虽然用函数 view 对三维视角进行控制十分方便，但其功能却非常有限。为了能够对三维场景进行全面控制，我们需要用到摄像头功能。当摄影师用一个摄像机拍摄电影时，必须要了解摄像机的全部功能，那么，当用户利用计算机设置三维图形或控制台游戏环境时，也必须了解摄像机的所有功能。在上述环境中，用户通常需要对两个三维坐标系统进行管理——一个是摄像机所在的坐标系，另一个是摄像机所指的坐标系，也就是摄像机目标坐标系。Matlab 提供了一些摄像机函数用于管理和处理这两个坐标系统之间的关系，并且提供对摄像机镜头的控制。

对于初学者而言，Matlab 摄像机函数的使用并不是一件容易的事情。因此，为了简化这些函数的使用，Matlab 将大多数的函数以 Tools 菜单或图标工具栏的形式提供给用户。利用这些交互式的摄像机工具，用户就可以避免在命令窗口中输入大量的函数和输入输出参数。考虑到摄像机函数描述起来比较复杂，并且这些函数大多都可以使用交互式工具完成，另外，真正深入使用这些函数的用户相对较少，因此本书没有对摄像机函数进行详细描述。Matlab 在线文档中有对这些函数详细的讨论。本章的最后也列出了 Matlab 提供的摄像机函数。

27.8 等高线图

等高线用于绘制具有相同海拔或高度的曲线。我们通常见到的地形图，就是用等高线绘制的地图。在 Matlab 中，二维等高线图和三维等高线图分别由函数 contour 和 countour3 绘制，如下例所示：

```
>> [X,Y,Z] = peaks;
>> subplot(1,2,1)
>> contour(X,Y,Z,20)          % generate 20 2-D contour lines
>> axis square
>> xlabel('X-axis'),ylabel('Y-axis')
>> title ('Figure 27.20a: 2-D Contour Plot')
>> subplot(1,2,2)
>> contour3(X,Y,Z,20)         % the same contour plot in 3-D
>> xlable('X-axis'),ylabel('Y-axis'),zlabel('Z-axis')
>> title('Figure 27.20b: 3-D Contour Plot')
```

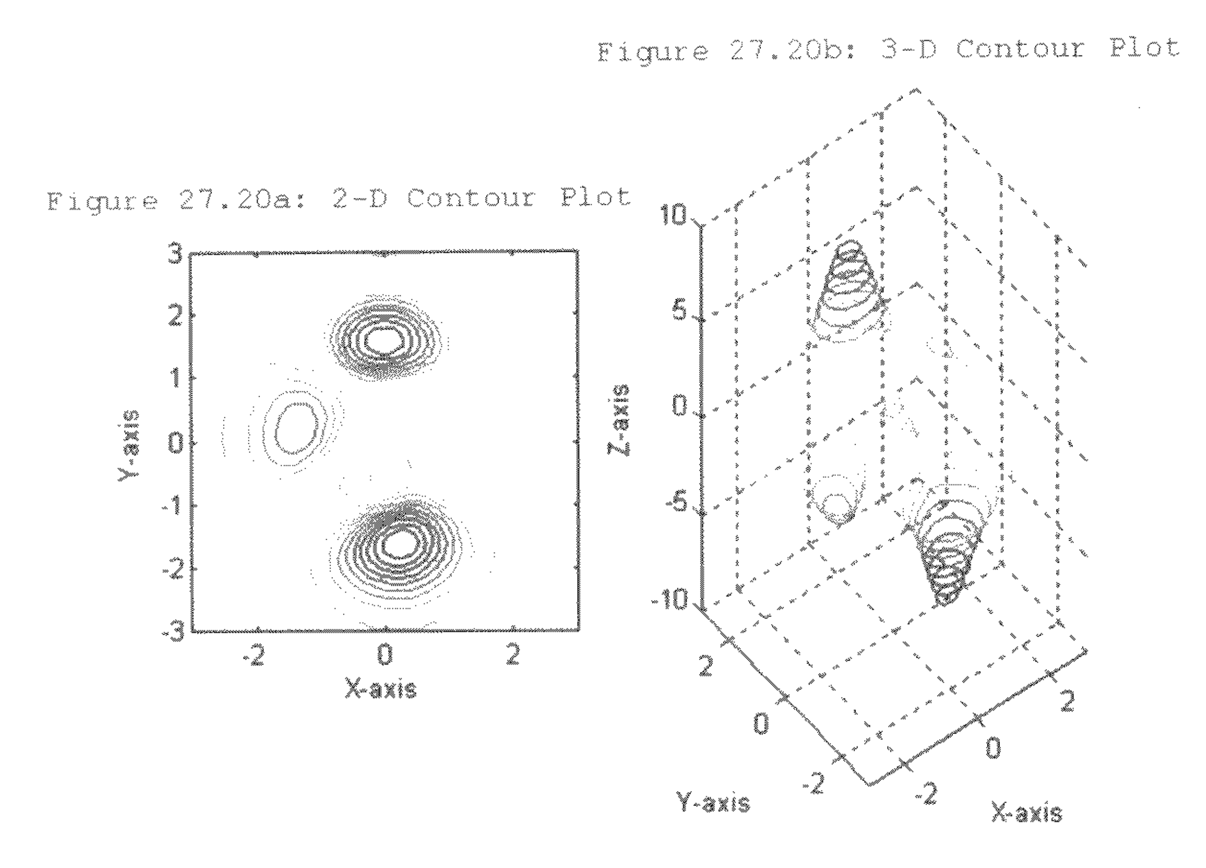


图 27.20 二维和三维等高线图

pcolor 函数用不同的颜色代表不同的高度来绘制等高线图，如下例所示：

```
>> subplot(1,2,1)
>> pcolor(X,Y,Z)
>> shading interp  % remove the grid lines
>> axis square
>> title('Figure 27.21a: Pseudocolor Plot')
```

有时用户还希望在图 27.21a 中的伪色图中添加二维等高线，以生成一幅填充了颜色的等高线图。在 Matlab 中，这种图形可以用函数 contourf 生成的，如下例所示：

```
>> subplot(1,2,2)
>> contourf(X,Y,Z,12)          % filled contour plot with 12 contours
>> axis square
>> xlabel('X-axis'), ylabel('Y-axis')
>> title('Figure 27.21b: Filled Contour Plot')
```

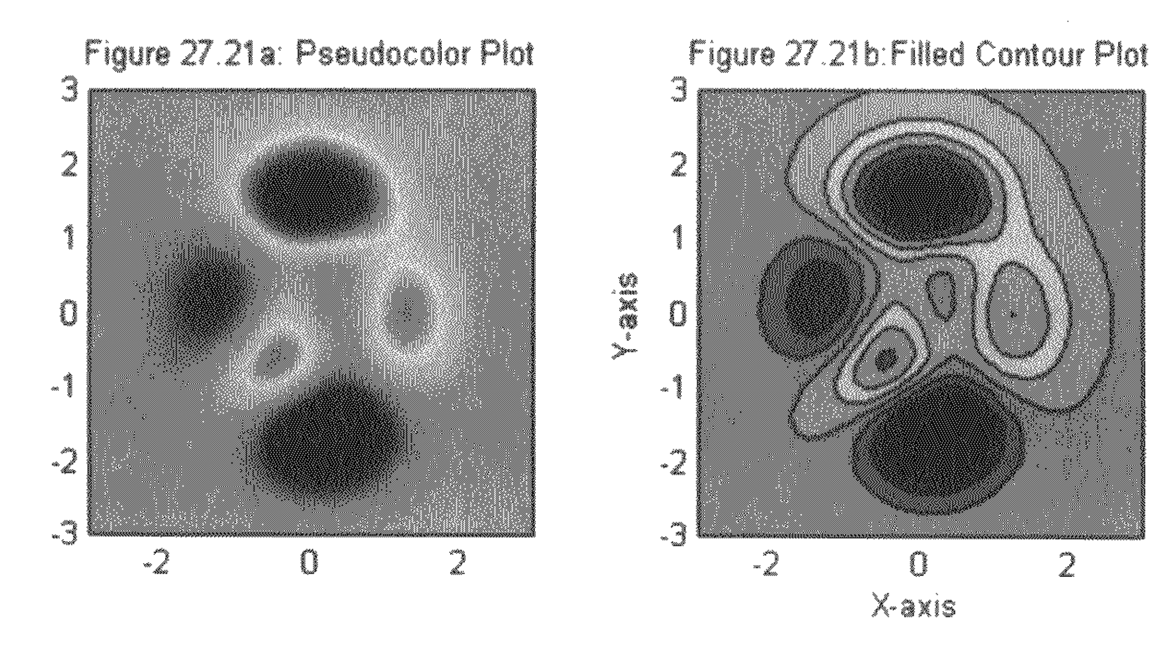


图 27.21 伪彩色图和彩色等高线图

我们也可以使用 clabel 函数对等高线进行标注。在标注时，clabel 函数需要函数 contour、contourf 和 contour3 返回的代表各个等高线的矩阵和（或）一个代表该图形的字符串标识，如下例所示：

```
>> C = contour(X,Y,Z,12);
>> clabel(C)
>> title('Figure 27.22: Contour Plot With Labels')
```

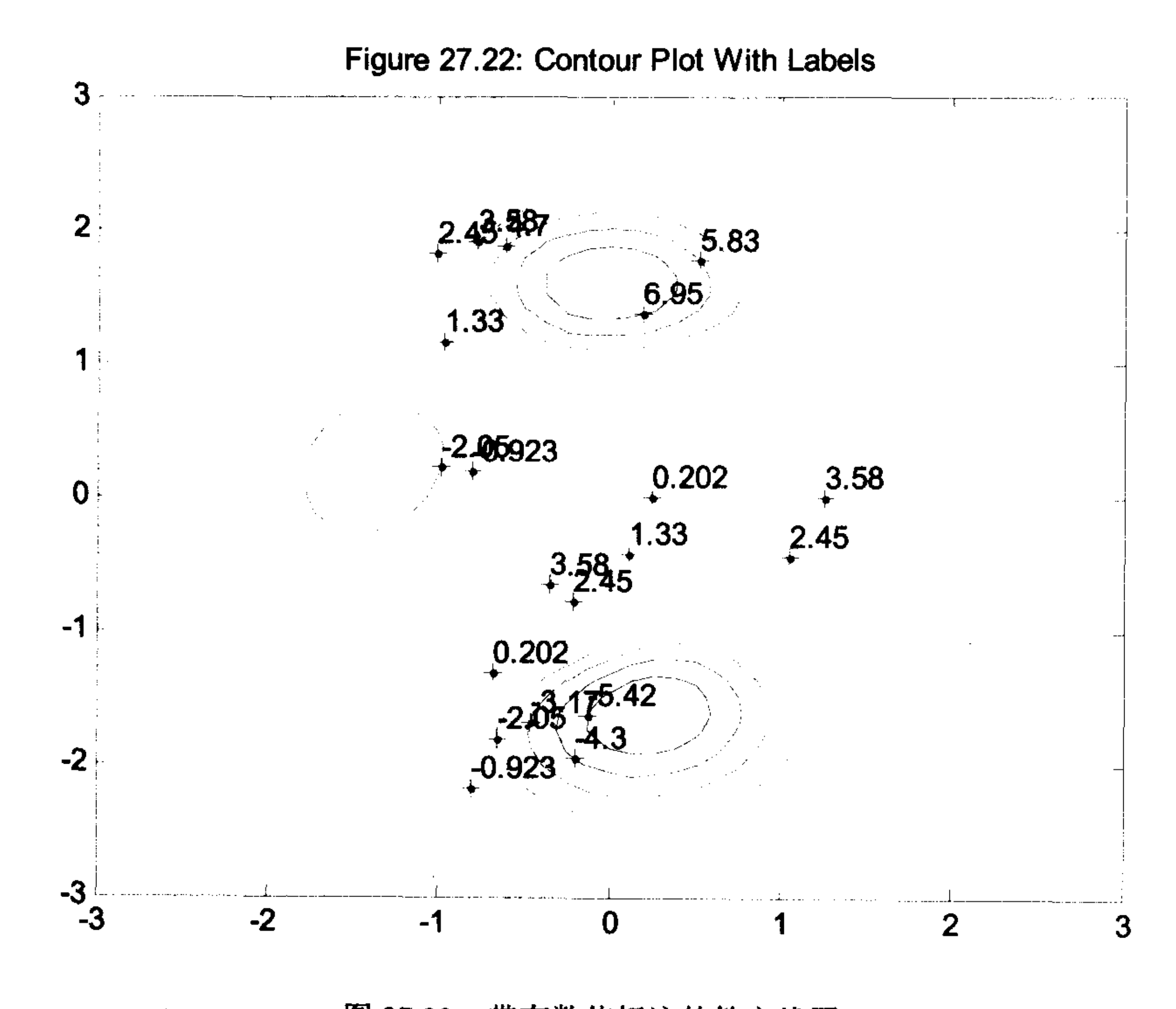


图 27.22　带有数值标注的等高线图

另外，使用等高线函数返回的两个参数就可以生成嵌入在等高线内部的标注。比如，将上面的第一行代码改为：[C,h] = contour(X,Y,Z,12)；clabel(C,h)，就可以产生与等高线同时出现的内部标注。最后，如果在 clabel 函数调用时，将'manual'作为最后一个可选输入参数提供给它，用户就可以用鼠标选择给哪条等高线添加标注。

27.9　特殊三维图形

除了前面讨论的绘图函数之外，Matlab 还提供了一些专用的三维绘图函数。例如，函数 ribbon(Y)用于将数组 Y 的各列画成一个个的带状；ribbon(x,Y)则绘制 Y 关于 x 的带状列图；ribbon(x,Y,width)则利用 width 参数设置各个带的宽度，如果用户不指定该参数，则默认的带宽度为 0.75。下例给出了 ribbon 函数的第一个用法：

```
>> Z = peaks;
>> ribbon(Z)
>> title('Figure 27.23: Ribbon Plot of Peaks')
```

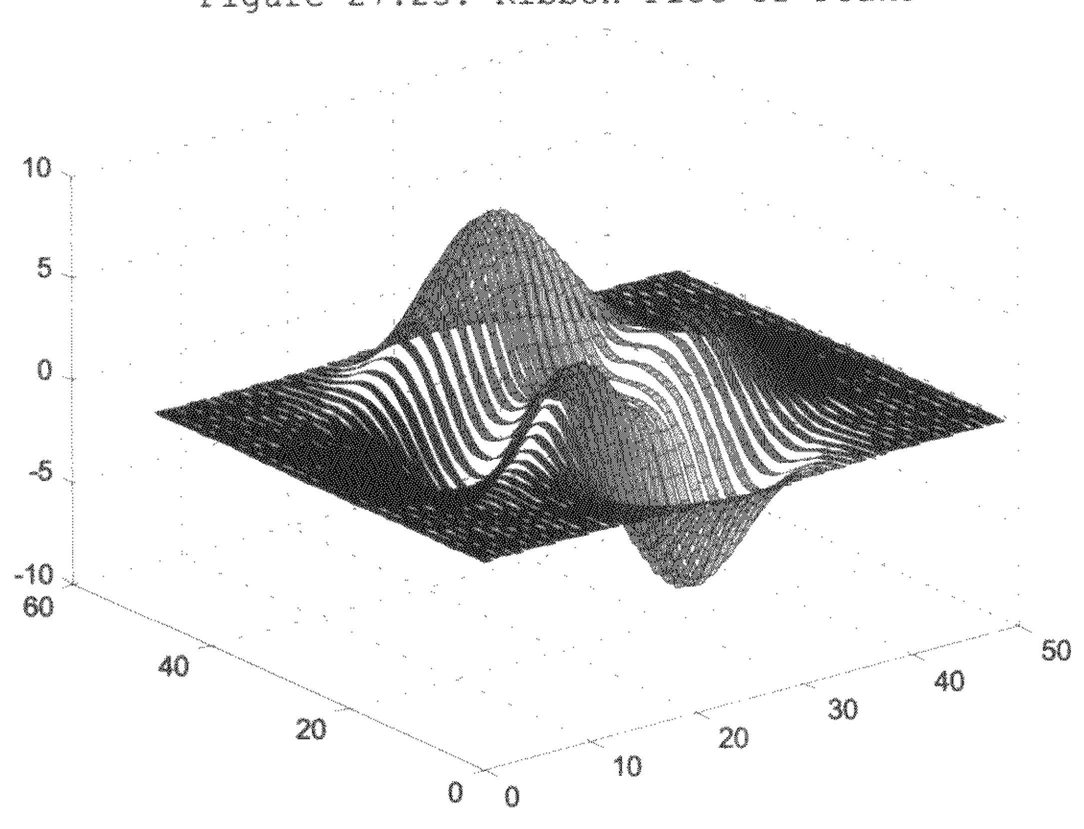


图 27.23 三维带状图

quiver(x,y,dx,dy) 函数用于在点(x, y)处画出方向或速度向量(dx, dy)（有些类似于前面绘制的法线向量），如下例所示：

```
>> [X,Y,Z] = peaks(16);
>> [DX,DY] = gradient(Z,.5,.5);
>> contour(X,Y,Z,10)
>> hold on
>> quiver(X,Y,DX,DY)
>> hold off
>> title('Figure 27.24:2-D Quiver Plot')
```

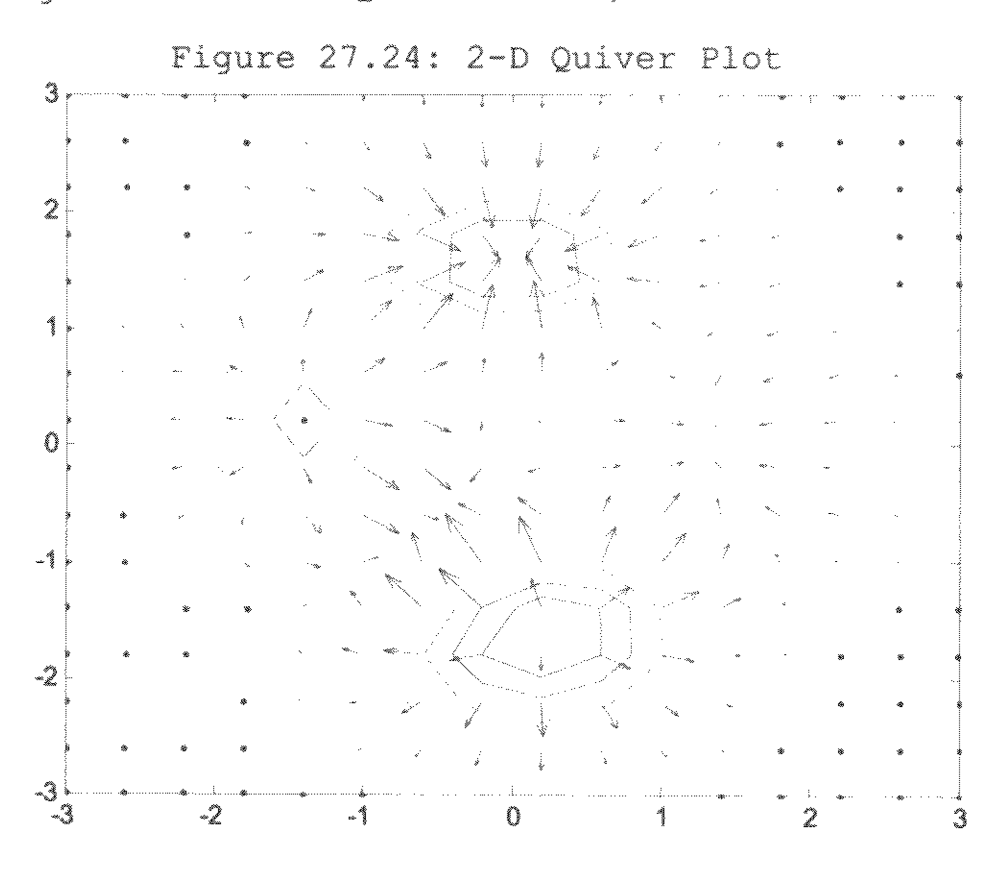


图 27.24 二维箭头图

上面的 quiver 函数只能绘制二维图形中的向量，quiver3(x,y,z,Nx,Ny,Nz)函数则用于绘制点(x,y,z)处的三维向量(Nx,Ny,Nz)，如下例所示：

```
>> [X,Y,Z] = peaks(20);
>> [Nx,Ny,Nz] = surfnorm(X,Y,Z);
>> surf(X,Y,Z)
>> hold on
>> quiver3(X,Y,Z,Nx,Ny,Nz)
>> axis tight
>> hold off
>> title('Figure 27.25: 3-D Quiver Plot')
```

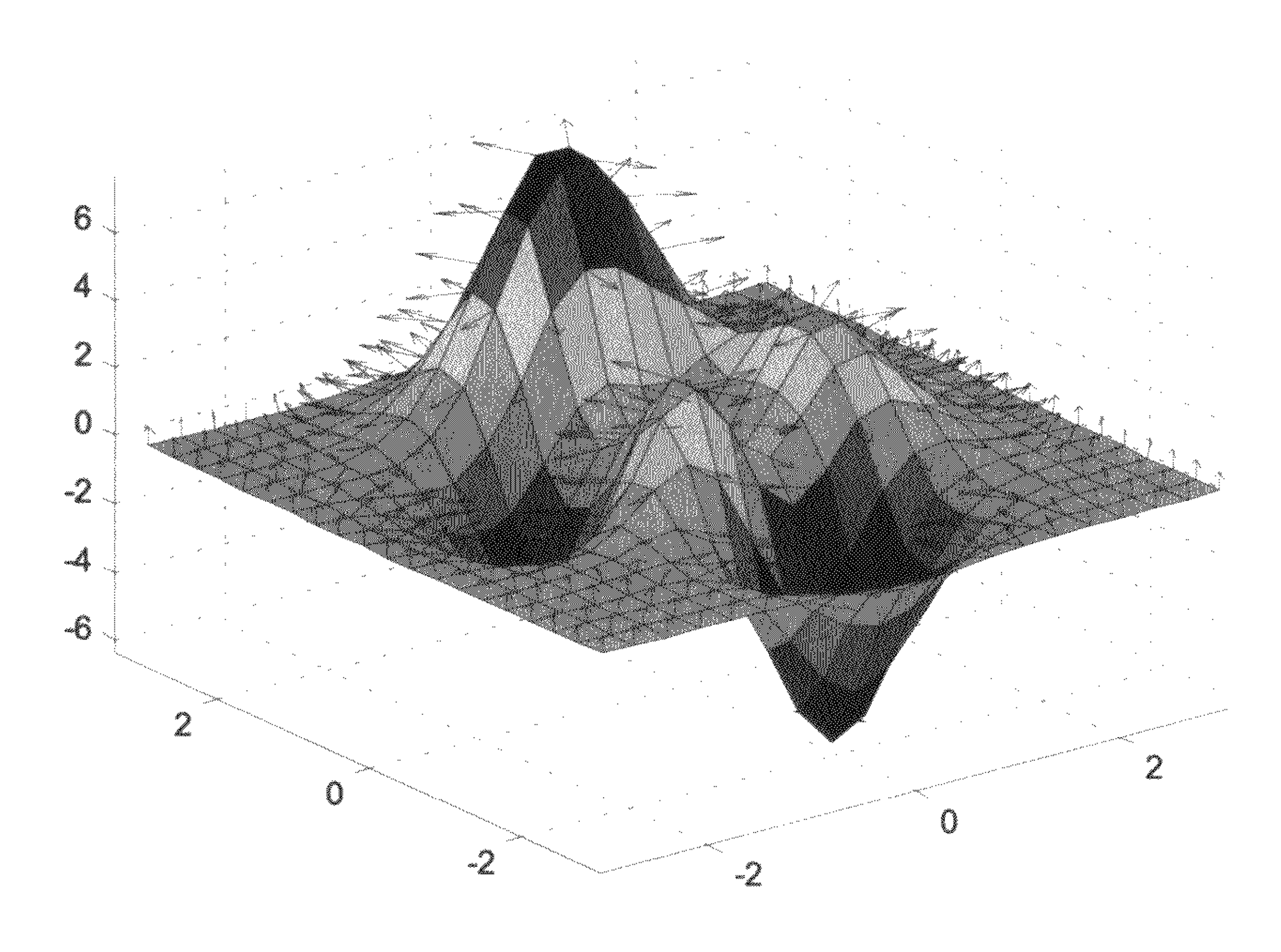
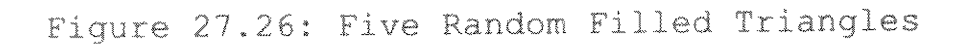


图 27.25　三维箭头图

fill3 为三维颜色填充函数，它是 fill 函数在三维中的等效函数。fill3 的调用方式为：fill3(X,Y,Z,C)，它用数组 X、Y 和 Z 作为顶点创建多个三角平面，并用 C 声明的颜色填充这些面。例如，下面的代码创建了 5 个随机大小的三角面，并用随机的颜色填充这些面：

```
>> fill3(rand(3,5),rand(3,5),rand(3,5),rand(3,5))
>> grid on
>> title('Figure 27.26: Five Random Filled Triangles')
```



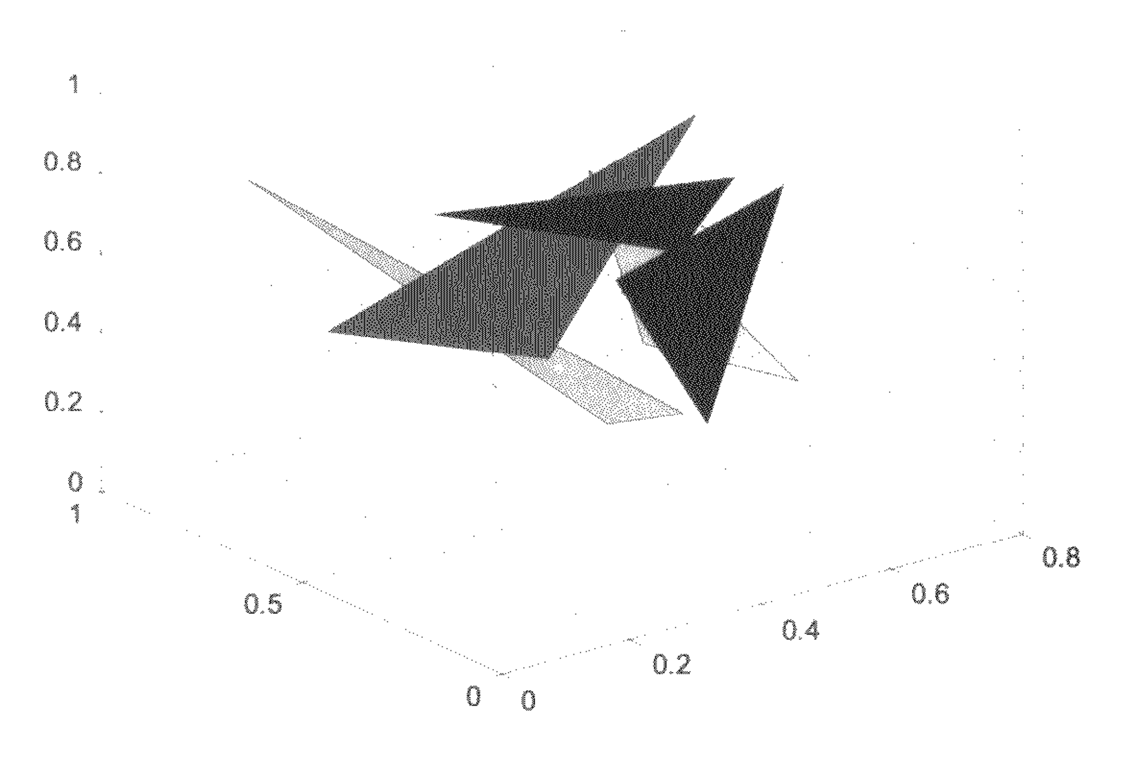


图 27.26　5 个随机填充的三角形

stem 函数在三维空间中也有等效函数，即 fill3，用于绘制三维空间中的离散数据序列。例如，stem3(X,Y,Z,C, 'filled')用由 x–y 平面出发的线条以及一个圆圈标记绘制(X,Y,Z)指定的各个点，其中可选参数 C 用于指定线条和标记的颜色，'filled'用于指定是否为标记填充颜色。如果只给 stem3 传递一个二维数组参数 Z，则该函数将绘制出 Z 中的各点，并且自动以 Z 的行和列作为 X 轴和 Y 轴，如下例所示：

```
>> Z = rand(5);
>> stem3(Z,'ro','filled');
>> grid on
>> title('Figure 27.27: Stem Plot of Random Data')
```

Figure 27.27: Stem Plot of Random Data

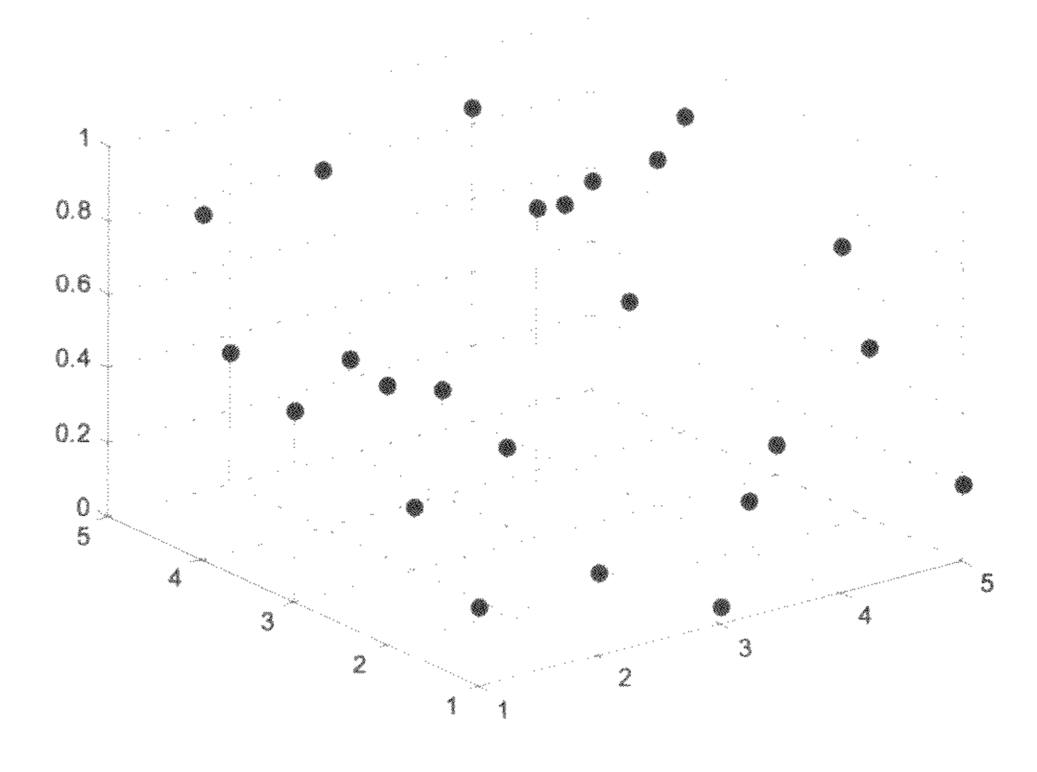


图 27.27　随机分布数据的三维柄状图

27.10 立体可视化

除了前面介绍的常用网格图、表面图和等高线图外，Matlab 还提供一些立体可视化函数用于绘制更为复杂的立体和向量对象。这些函数通常在三维空间中构建标量和向量的图形。由于这些函数构建的是立体而不是一个简单的表面，因此它们需要三维数组作为输入参数，其中三维数组的每一维分别代表一个坐标轴，三维数组中的点定义了坐标轴栅格和坐标轴上的坐标点。如果要绘制的函数是一个标量函数，则绘图函数需要 4 个三维数组，其中 3 个数组各代表一个坐标轴，第四个数组代表了这些坐标处的标量数据，这些数组通常记作 X、Y、Z 和 V。如果要绘制的函数是一个向量函数，则绘图函数需要 6 个三维数组，其中 3 个各表示一个坐标轴，3 个用来表示坐标点处的向量，这些数组通常记作 X、Y、Z、U、V 和 W。

要正确合理地使用 Matlab 提供的立体和向量可视化函数，用户需要对与立体和向量有关的一些术语有所了解。比如，散度（divergence）和旋度（curl）用于描述向量过程，而等值面（isosurfaces）和等值顶（isocaps）则用于描述立体的视觉外观。如果用户要生成和处理比较复杂的立体对象，就需要参考相应的文献对这些术语进行深入了解。不过，本书并不详细讲述这些术语的具体含义，而只是通过几个简单的例子讲述 Matlab 中如何利用数据数组创建立体结构。关于这方面更详细的信息请读者参考相应的 Matlab 在线帮助文档。

下面我们看一个利用标量函数构建立体图形的例子。首先，我们必须生成一个构建立体对象的坐标系，代码如下：

```
>> x = linspace(-3,3,13);        % x-coordinate points
>> y = 1:20;                     % y-coordinate points
>> z = -5:5;                     % Z-coordinate points
>> [X,Y,Z] = meshgrid(x,y,z);    % meshgrid works here too!
>> size(X)
ans =
    20      13      11
```

上面的代码演示了 meshgrid 函数在三维空间中的应用。其中，X、Y、Z 为定义栅格的 3 个三维数组。这三个数组分别是从 x、y 和 z 经过三维栅格扩展形成的。也就是说，X 是将 x 复制扩展成具有 length(y)个行和 length(z)个页的三维数组，Y 是先将 y 转置成一个列向量，然后复制扩展成具有 length(x)个列和 length(z)个页的三维数组，Z 则是先将 z 转换成一个 1×1×length(z)的三维向量，然后复制扩展成具有 length(y)个行和 length(x)个列的三维数组。

然后，我们还需要定义一个以这三个数组为自变量的标量函数 V，代码如下：

```
>> V = sqrt(X.^2 + cos(Y).^2 + Z.^2);
```

这样，利用标量函数 $v=f(x,y,z)$定义一个立体对象所需要的数据已全部给出。为了使该立体对象可视化，我们可以利用下面的代码查看该立体对象的一些截面：

```
>> slice(X,Y,Z,V,[0 3],[5 15],[-3 5])
>> xlabel('X-axis')
>> ylabel('Y-axis')
```

```
>> zlabel('Z-axis')
>> title('Figure 27.28: Slice Plot Through a Volume')
```

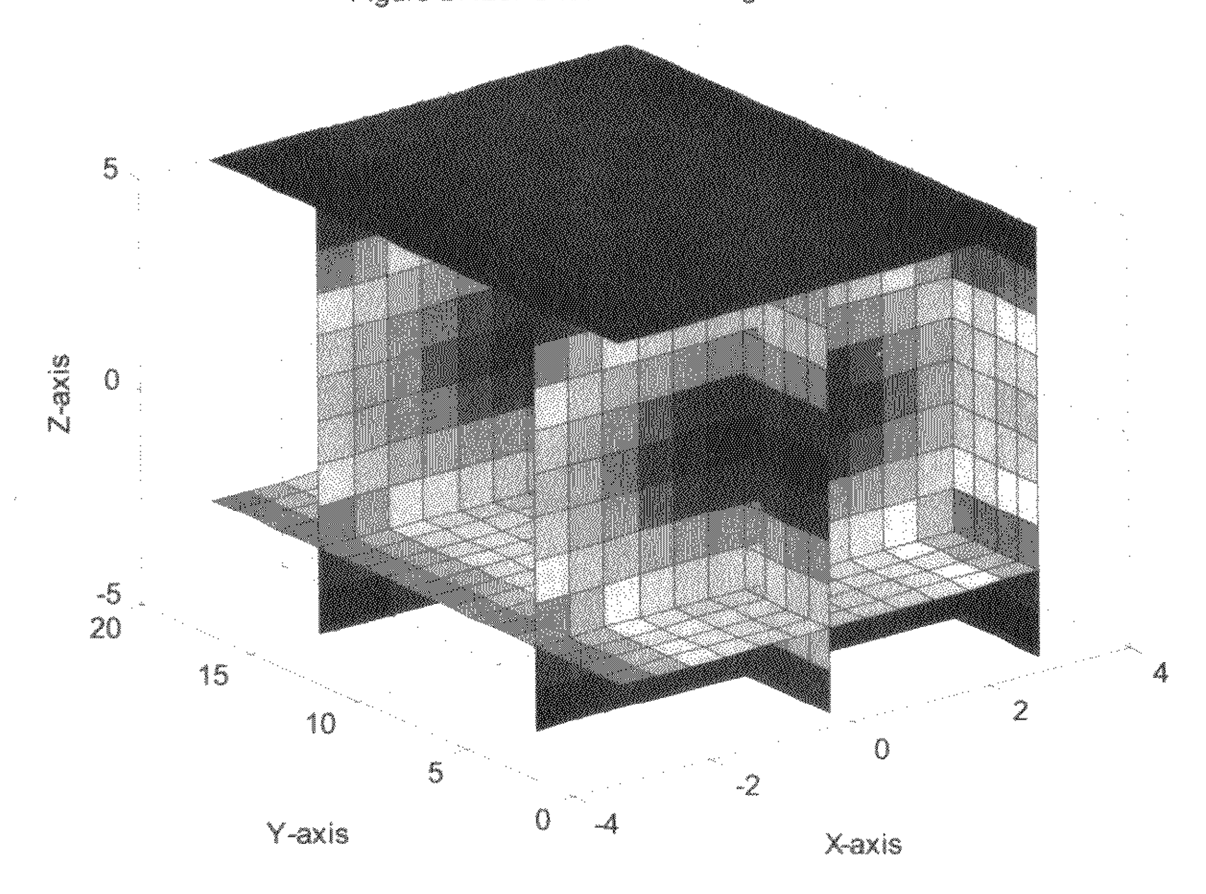


图 27.28 立体截面图

图 27.28 显示了立体图形在 x=0、x=3、y=5、y=15、z=−3 和 z=5 所定义的平面上的截面。图中的颜色是根据截面上的 V 值来进行绘制的。

图 27.28 中演示了立体图形的平面截面，在立体图形中，也可以显示立体图形的曲面截面。例如，下面的代码采用正弦函数来截取立体图形的截面，其中，xs、ys 和 zs 定义了一个截取立体图形的一个正弦截面：

```
>> [xs,ys] = meshgrid(x,y);
>> zs = sin(-xs+ys/2);             % a surface to use
>> slice(X,Y,Z,V,xs,ys,zs)
>> xlabel('X-axis')
>> ylabel('Y-axis')
>> zlabel('Z-axis')
>> title('Figure 27.29: Slice Plot Using a Surface')
```

除了截取平面以外，用户还可以用 contourslice 函数为截取的平面添加等高线，如下例所示：

```
>> slice(X,Y,Z,V,[0 3],[5 15],[-3 5])
>> hold on
>> h = contourslice(X,Y,Z,V,3,[5 15],[]);
>> set(h,'EdgeColor','k','Linewidth',1.5)
>> xlabel('X-axis')
>> ylabel('Y-axis')
>> zlabel('Z-axis')
>> title('Figure 27.30: Slice Plot with Selected Contours')
>> hold off
```

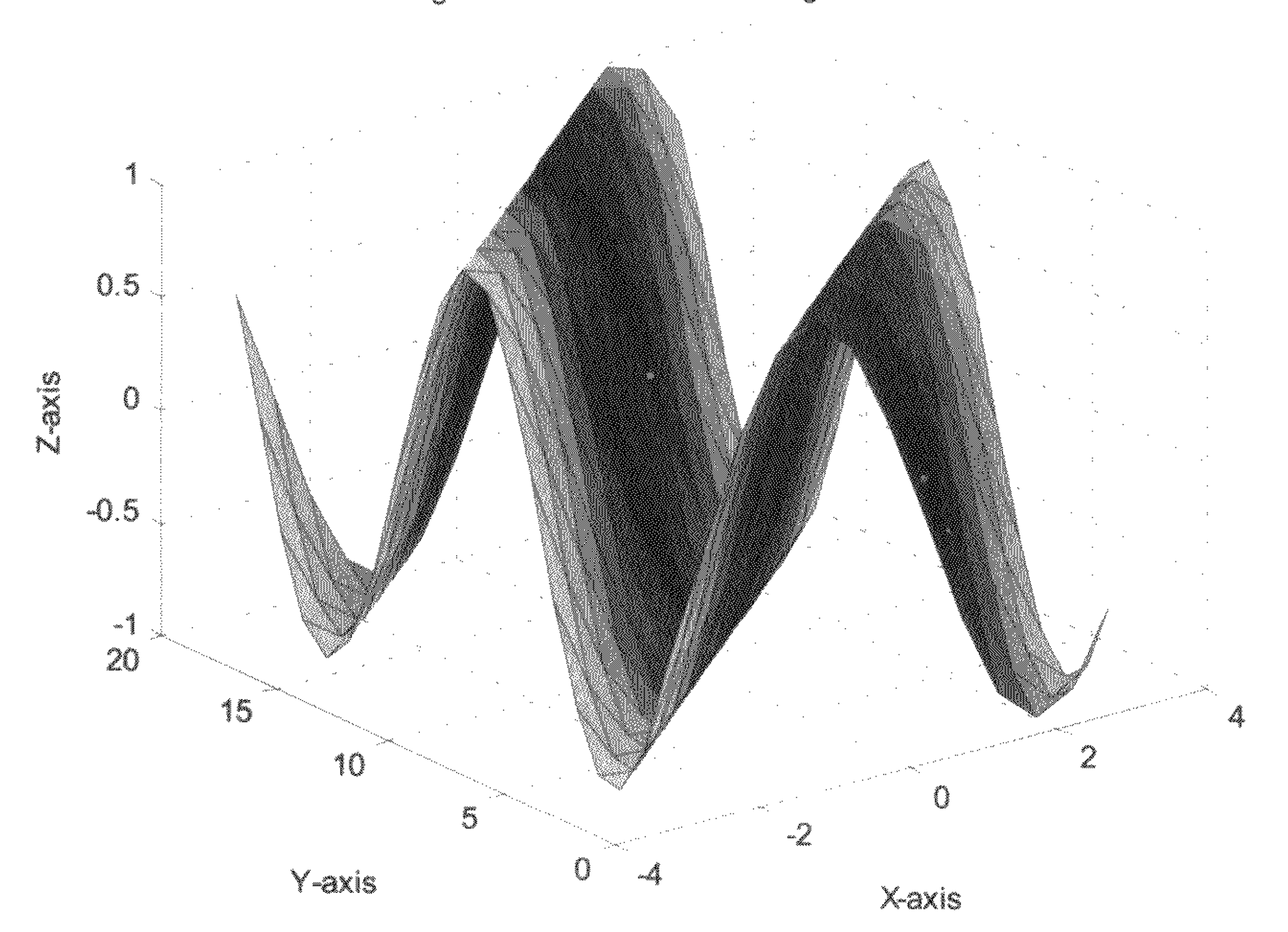


图 27.29 使用正弦曲面截取立体图形

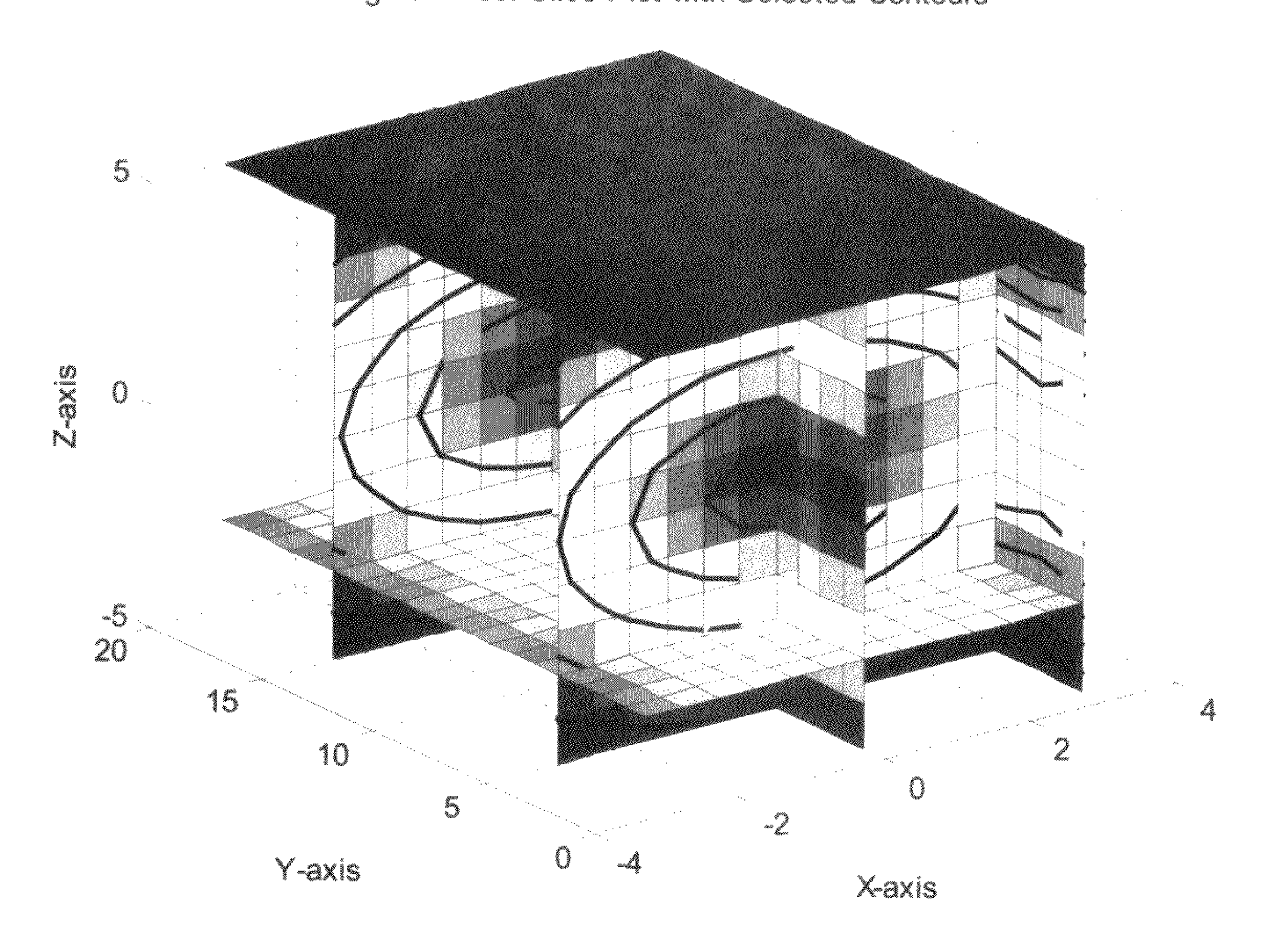


图 27.30 在立体图形的截面上绘制等高线

图 27.30 中，等高线被分别添加到了 x=3、y=5 和 y=15 的截面上，并利用句柄图形函数 set 将其颜色设置为黑色，宽度设置为 1.5 个像素点。

除了查看立体对象的截面之外，寻找使 V 等于某个特定值的表面（称为等值面）也十分常见。在 Matlab 中，这一操作可以用 isosurface 函数实现，该函数与 Delaunay 函数类似，也返回若干三角形的顶点。将 isosurface 函数的返回结果传递给 patch 函数，则可以绘制出由这些三角形构成的等值面。下面给出了一个绘制等值面的例子：

```
>>[X,Y,Z,V] = flow(13);              % get flow data
>> fv = isosurface(X,Y,Z,V,-2);     % find surface of value -2

>> subplot(1,2,1)
>> p= patch(fv);                     % plot V = -2 surface
>> set(p,'FaceColor',[.5 .5 .5],'EdgeColor','Black');     % modify patches
>> view(3), axis equal tight, grid on  % pretty it up
>> title({'Figure 27.31a:' 'Isosurface Plot, V = 2'})

>> subplot(1,2,2)
>> p = patch(shrinkfaces(fv,.3));        % shrink faces to 30% of original
>> set(p, 'Facecolor',[.5 .5 .5], 'EdgeColor', 'Black');% modify patches
>> view(3),axis equal tight,grid on     % pretty it up
>> title({'Figure 27.31b:' ' Shrunken Face Isosurface Plot, V = 2'})
```

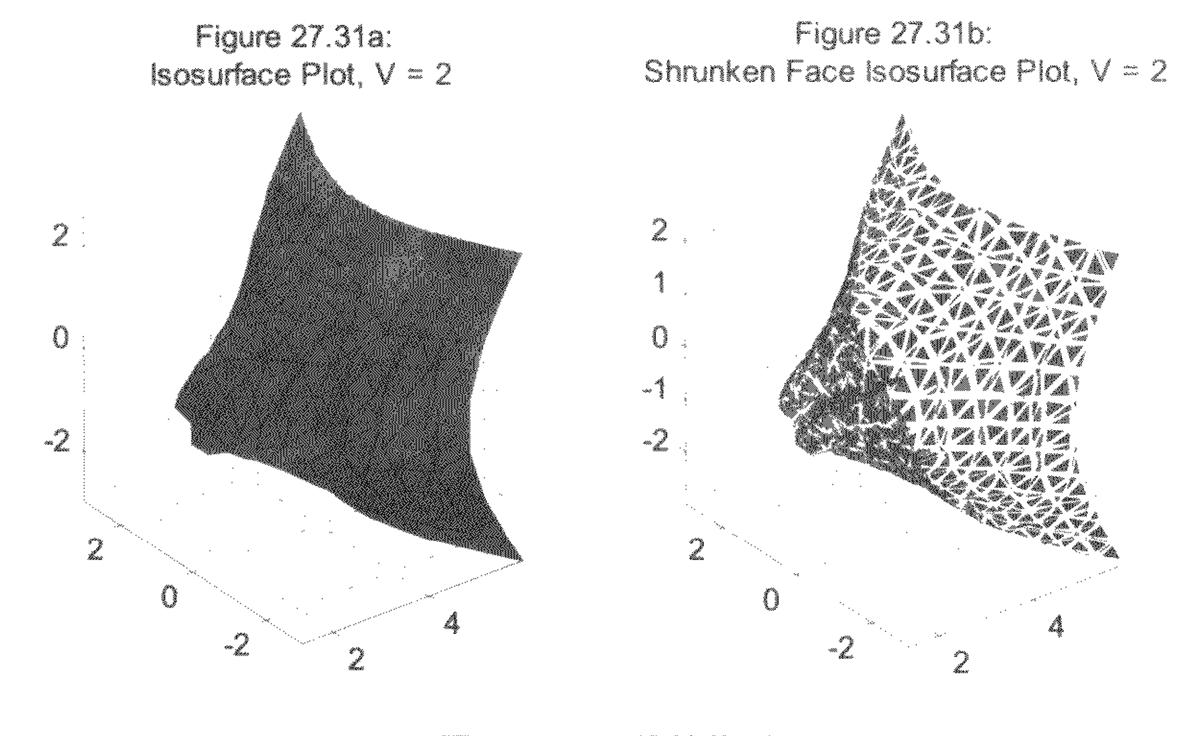


图 27.31 三维等值面

图 27.31b 还展示了函数 shrinkfaces 的用法，顾名思义，该函数的功能为使表面收缩。

当我们显示立体图形仅仅是为了观察其大体结构时，就没有必要针对所有的数据点作图，因为数据点太多，会降低显示的速度。利用函数 reducevolume 和 reducepatch 则可以使用户在显示图形之前先删除一些数据或一些对图形显示影响很小的碎片，从而提高图形显示的效率。下面的例子演示了这两个函数的具体用法：

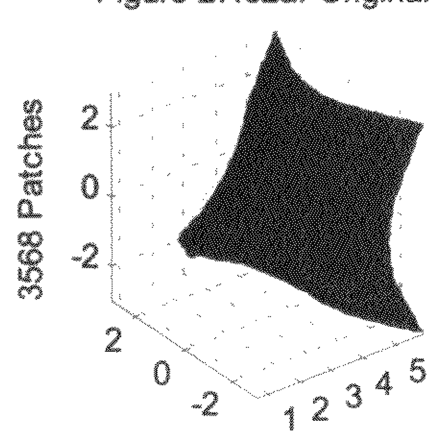
```
>> [X,Y,Z,V] = flow;
>> fv = isosurface(X,Y,Z,V,-2);
>> subplot(2,2,1)    % Original
>> p = patch(fv);
>> Np = size(get(p,'Faces'),1);
>> set(p,'FaceColor',[.5 .5 .5],'EdgeColor','Black');
>> view(3),axis equal tight,grid on      % pretty it up
```

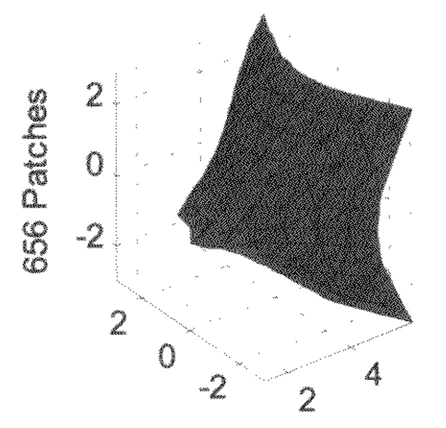
```
>> zlabel(sprintf('%d Patches',Np))
>> title('Figure 27.32a: Original')

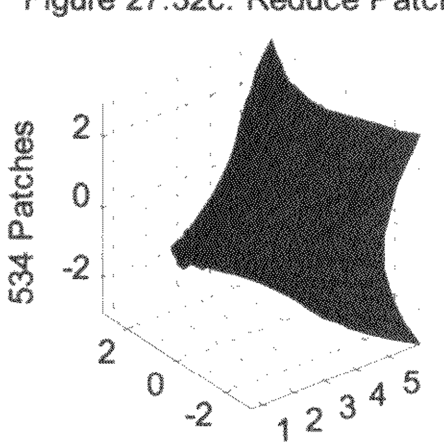
>> subplot(2,2,2)   % Reduce Volume
>> [Xr,Yr,Zr,Vr] = reducevolume(X,Y,Z,V,[3 2 2]);
>> fvr = isosurface(Xr,Yr,Zr,Vr,-2);
>> p = patch(fvr);
>> Np = size(get(p,'Faces'),1);
>> set(p,'FaceColor',[.5 .5 .5],'EdgeColor','Black');
>> view(3),axis equal tight,grid on    % pretty it up
>> zlabel(sprintf('%d Patches',Np))
>> title('Figure 27.32b: Reduce Volume')

>> subplot(2,2,3)   % Reduce Patch
>> p = patch(fv);
>> set(p,'FaceColor',[.5 .5 .5],'EdgeColor','Black');
>> view(3),axis equal tight,grid on    % pretty it up
>> reducepatch(p,.15)   % keep 15 percent of the faces
>> Np = size(get(p,'Faces'),1);
>> zlabel(sprintf('%d Patches',Np))
>> title('Figure 27.32c: Reduce Patches')

>> subplot(2,2,4)   % Reduce Volume and Patch
>> p = patch(fvr);
>> set(p,'FaceColor',[.5 .5 .5],'EdgeColor','Black');
>> view(3),axis equal tight,grid on    % pretty it up
>> reducepatch(p,.15)   % keep 15 percent of the faces
>> Np = size(get(p,'Faces'),1);
>> zlabel(sprintf('%d Patches',Np))
>> title('Figure 27.32d: Reduce Both')
```







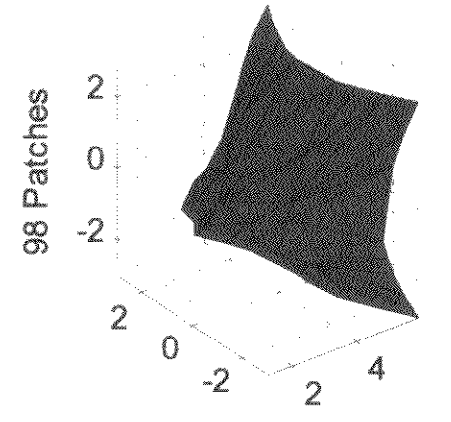


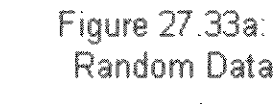
图 27.32　利用不同大小的碎片绘制三维表面

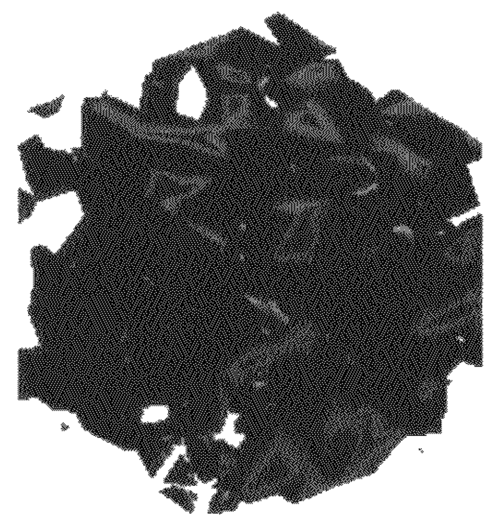
三维数据也可以通过用 smooth3 函数来过滤而实现其平滑化，例如：

```
>> data = rand(10,10,10);              % random data
>> datas = smooth3(data,'box',3);      % smoothed data

>> subplot(1,2,1)   % random data
>> p = patch(isosurface(data,.5),...
    'FaceColor','Blue','EdgeColor','none');
>> patch(isocaps(data,.5),...
    'FaceColor','interp','EdgeColor','none');
>> isonormals(data,p)
>> view(3); axis vis3d tight off
>> camlight; lighting phong
>> title({'Figure 27.33a:' ' Random Data'})

>> subplot(1,2,2)   % smoothed random data
>> p = patch(isosurface(datas,.5),...
    'FaceColor','Blue','EdgeColor','none');
>> patch(isocaps(datas,.5),...
    'FaceColor', 'interp', 'EdgeColor', 'none');
>> isonormals(datas,p)
>> view(3); axis vis3d tight off
>> camlight; lighting phong
>> title({'Figure 27.33b:' ' Smoothed Data'})
```





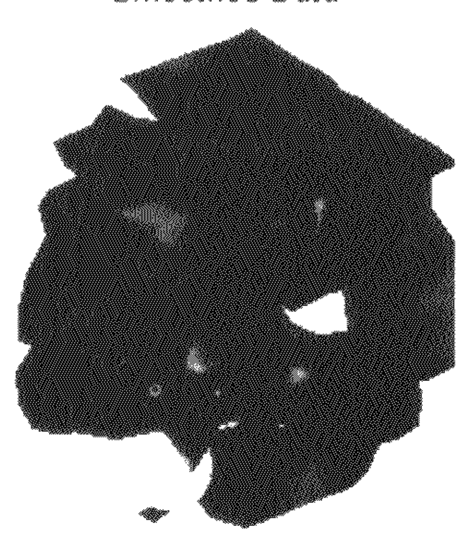


图 27.33 三维数据平滑

上边的例子展示了函数 isocaps 和 isonormals 的用法。函数 isocaps 生成块状图的外层表面。函数 isonormals 调整所画碎片的属性，使得所显示的图形有正确的光照效果。

27.11 轻松绘图

当用户不想花费时间来显式地声明一个三维图形数据点的时候，Matlab 提供了函数 ezcontour、ezcontour3、ezmesh、ezmeshc、ezplot3、ezsurf 和 ezsurfc。这些函数构建的图形类似于它们不带 ez 前缀的等价函数所构建的图形。但是，它们的输入参数是函数，这些函

数是由字符串或符号数学对象，以及可选的图形所在的坐标轴限所定义的。在这些函数内部，它们对数据进行计算，然后生成想要的图形，例如：

```
>> fstr = ['3*(1-x).^2.*exp(-(x.^2) - (y+1).^2)' ...
  ' - 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2)' ...
  ' - 1/3*exp(-(x+1).^2 - y.^2)'];

>> subplot(2,2,1)
>> ezmesh(fstr)
>> title('Figure 27.34a: Mesh of peaks(x,y)')
>> subplot(2,2,2)
>> ezsurf(fstr)
>> title('Figure 27.34b: Surf of peaks(x,y)')
>> subplot(2,2,3)
>> ezcontour(fstr)
>> title('Figure 27.34c: Contour of peaks(x,y)')
>> subplot(2,2,4)
>> ezcontourf(fstr)
>> title('Figure 27.34d: Contourf of peaks(x,y)')
```

Figure 27.34a: Mesh of peaks(x,y)

Figure 27.34b: Surf of peaks(x,y)

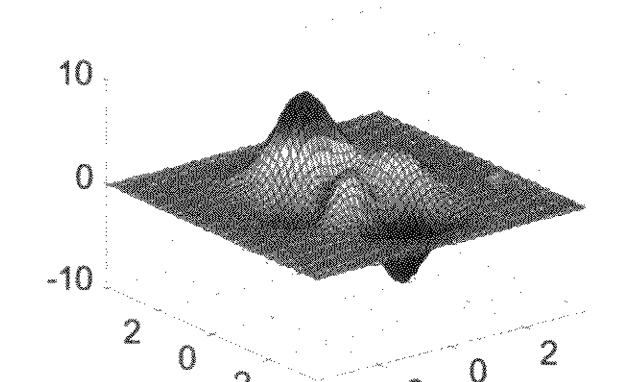


Figure 27.34c: Contour of peaks(x,y)

Figure 27.34d: Contourf of peaks(x,y)

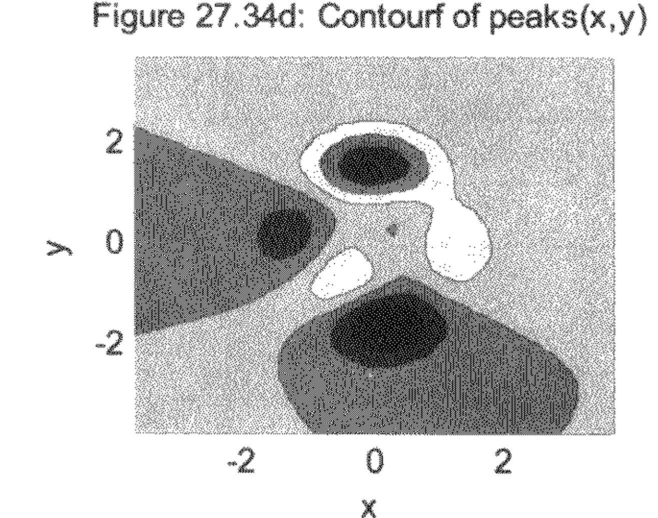


图 27.34 轻松绘图举例

27.12 小结

作为小结，下面的表格列出了 Matlab 中进行三维绘图的函数：

函数	描述
plot3	在三维空间绘制线和点
mesh	绘制网格表面
meshc	绘制底面带等高线的网格图
meshz	绘制带零平面的网格图
surf	绘制表面图
surfc	绘制底面带等高线图的表面图
surfl	绘制有基本光照属性的表面图
fill3	绘制填充颜色的三维多面体
shading	设置颜色投影模式
hidden	显示或隐藏被网格遮住的部分
surfnorm	绘制表面法线
axis	控制坐标轴刻度和外观
grid	显示或隐藏栅格线
box	显示或隐藏坐标轴边框
hold	保持当前的图形
subplot	在同一图形窗口中生成多个坐标轴
daspect	设定数据高宽比
pbaspect	设置屏幕高宽比
xlim	设置 x 坐标轴范围
ylim	设置 y 坐标轴范围
zlim	设置 z 坐标轴范围
view	指定三维视角
viewmtx	视角转换矩阵
rotate3d	交互式坐标轴旋转
campos	摄像头位置
camtarget	摄像头目标
camva	摄像头视角
camup	设置窗口相对于显示对象的位置向量
camproj	摄像头投影
camorbit	摄像头轨迹
campan	固定窗口位置旋转对象
camdolly	固定对象移动摄像头
camzoom	放大摄像头
camroll	滚动摄像头
camlookat	查找特定的对象
camlight	生成摄像头光照对象并将其放置在合适的位置

（续表）

函数	描述
title	为图形添加标题
xlabel	添加 x 轴标注
ylabel	添加 y 轴标注
zlabel	添加 z 轴标注
text	在图形中放置文本
gtext	在鼠标点击的位置放置文本
contour	绘制等高线图
contourf	填充等高线图
contour3	绘制三维等高线图
clabel	添加等高线标注
pcolor	绘制伪色图
voronoi	绘制 Voronoi 分割图
trimesh	绘制三角形网格图
trisurf	绘制三角形表面图
scatter3	绘制三维分散数据图
stem3	绘制三维柄状图
waterfall	绘制瀑布图
ezmesh	利用字符串表达式轻松绘制网格图
ezmeshc	利用字符串表达式轻松绘制带等高线的网格图
ezplot3	利用字符串表达式轻松绘制三维曲线图
ezsurf	利用字符串表达式轻松绘制表面图
ezsurfc	利用字符串表达式轻松绘制带等高线的表面图
ezcontour	利用字符串表达式轻松绘制等高线图
ezcontourf	利用字符串表达式轻松绘制填充了颜色的等高线图
vissuite	有关可视化的帮助文档
isosurface	从立体图形中截取等值面
isonormals	绘制等值面法线
isocaps	绘制边缘等值面
isocolors	设置等值面和碎片颜色
contourslice	在截面中绘制等高线
slice	在立体图中截取曲面
streamline	数据的流线
stream3	三维数据流线
stream2	二维数据流线
quiver3	绘制三维箭头图

（续表）

函数	描述
quiver	绘制二维箭头图
divergence	一个向量域的散度
curl	一个向量域的曲度和角速度
coneplot	绘制锥形图
streamtube	绘制流形管状图
streamribbon	绘制流形带状图
streamslice	在截面中绘制流形线
streamparticles	显示流粒子
interpstreamspeed	根据速度值对流线顶点进行插值
subvolume	提取三维数据集合中的子集
reducevolume	减小三维数据集
volumebounds	返回体积和颜色范围
smooth3	三维数据平滑
reducepatch	在绘制时减少碎片的数量
shrinkfaces	在绘制时缩小碎片的大小

Chapter 28

使用颜色和光照

利用图形和图表来表示数据集的技术称为数据可视化。Matlab 提供了一系列工具和函数使二维或三维数据可视化，以便向用户提供更多的信息。比如，一条正弦曲线就比代表这条曲线的一组数据向用户提供了更多的信息。除了是一个功能强大的计算引擎之外，Matlab 在数据可视化方面也毫不逊色。

有时候，一个简单的二维或三维图形无法显示出用户需要的所有信息。这时，可以使用颜色来表示数据的第四维信息。前一章讨论的大多数绘图函数都接受一个颜色参数，用来表示额外增加的一维。

本章在介绍颜色参数之前，首先对颜色表进行简单描述，包括颜色表的创建、使用、显示以及修改等。然后，介绍了如何在一个图形窗口中仿真多个颜色表，或者使用一个颜色表的部分内容。最后，讨论了光照模型，并给出了一些光照模型的示例。

需要指出的是，与上一章的图形一样，本章所显示的图形也无法展示出颜色的变化，但读者在计算机屏幕上可以观察详细的颜色变化信息。因此，建议用户在学习本书时，将书中提供的代码输入到 Matlab 中进行验证，以便更好地理解本章所讲的内容。

28.1 理解颜色表

Matlab 使用一个列数为 3 的数值数组来表示颜色值，这个数组被称作颜色表。颜色表数组中的元素值均介于 0 和 1 之间，其中每一行元素代表一个不同的颜色，该行中的 3 个元素分别表示构成该行颜色的红、绿和蓝色的强度。下表给出了颜色表中部分数值与对应颜色之间的关系：

红	绿	蓝	颜色
1	0	0	红
0	1	0	绿
0	0	1	蓝
1	1	0	黄
1	0	1	洋红
0	1	1	青

（续表）

红	绿	蓝	颜色
0	0	0	黑
1	1	1	白
0.5	0.5	0.5	中灰
0.67	0	1	紫
1	0.4	0	橙
0.5	0	0	暗红
0	0.5	0	暗绿

上面的表格表明，颜色表中的第一列数据表示红色强度，第二列数据表示绿色强度，第三列数据表示蓝色强度。颜色表中所有的值都被严格限制在 0 和 1 之间。

一个颜色表由多个代表红-绿-蓝（RGB）值的行组成，不同的颜色表从第一行到最后一行按照不同的方式排列和变化。用户可以根据需要自定义颜色表，同时，Matlab 也提供了一些预先定义好的颜色表，如下表所示：

颜色表函数	描述
hsv	色度-饱和度-亮度（HSV）颜色表，以红开始，以红结束
jet	HSV 颜色表的变种，以蓝色开始，以红色结束
hot	按照黑色-红色-黄色-白色的顺序排列的颜色表
cool	暗青色和暗洋红颜色表
summer	暗绿色和暗黄色颜色表
autumn	暗红色和暗黄色颜色表
winter	暗绿色和暗蓝色颜色表
spring	暗洋红和暗黄色颜色表
white	全白颜色表
gray	线性灰度变化的颜色表
bone	基色为浅蓝色的灰度颜色表
pink	粉红色颜色表
copper	线性变化的铜色颜色表
prism	红色，橙色，黄色，绿色，蓝色和紫色交替出现的颜色表
flag	红色，白色，蓝色和黑色交替出现的颜色表
lines	交替绘制的线性颜色表
colorcube	增强的颜色立方体

在默认情况下，上表中的颜色表函数都自动生成一个 64×3 的数组，该数组给出了颜色表中 64 种颜色的 RGB 描述。当然，上表中的颜色表函数也都可以接受一个参数，声明需要生成颜色数组的行数。例如，hot(m)就生成一个 m×3 的数组，该数组包含了从黑色、暗红、橙色、黄色到白色共 m 种颜色的 RGB 值。

28.2　使用颜色表

语句 colormap(M)将数组 M 作为当前“图形”窗口使用的颜色表。例如，colormap(cool)就将上节表中的 cool 颜色表作为当前图形窗口使用的颜色表。colormap default 将默认安装的颜色表作为当前图形窗口的颜色表，通常是 hsv 或者 jet。

诸如 plot 和 plot3 这样的绘制直线或曲线的函数都没有用到颜色表，在表示线的颜色时，它们使用提供给它们的输入参数信息，或者使用一个默认的颜色顺序。因此，这些函数所用到的颜色会随着用户的不同选择而变化。除了这两个函数以外，其他大多数绘图函数，比如 mesh、surf、contour、fill、pcolor 等，都使用当前的颜色表来决定绘图时的颜色序列。

总之，一个函数通常以下列 3 种方式之一来接受一个颜色参数：

（1）一个在 plot 颜色和线型表中定义的字符串，例如，'r'或'red'表示红色。

（2）一个表示单个颜色的 RGB 行向量，例如，[.25 .50 .75]。

（3）一个数组。如果颜色参数是一个数组，那么 Matlab 就会对该数组的元素进行换算，使其作为对当前颜色表中颜色值的索引。

28.3　显示颜色表

颜色表的显示有多种方法。其中一种方法就是直接查看一个颜色表数组中的元素，例如：

```
>> hot(8)
ans =
        0.33333       0          0
        0.66667       0          0
        1             0          0
        1             0.33333    0
        1             0.66667    0
        1             1          0
        1             1          0.5
        1             1          1
>> gray(5)
ans =
       0             0          0
       0.25          0.25       0.25
       0.5           0.5        0.5
       0.75          0.75       0.75
       1             1          1
```

上面两个例子显示的都是标准的颜色表。其中第一个颜色表是含有 8 个颜色值的 hot 颜色表，第二个是含有 5 个颜色值的 gray 颜色表。gray 颜色表的 3 个颜色分量相等，并且通过等值增加每个颜色分量的值，得到了不同深度的灰色。

为了更清楚地看到颜色表中的颜色，需要将颜色表进行可视化显示。这时，用户可以使用函数 pcolor 和 rgbplot，如下例所示：

```
>> n = 21;
>> map = copper(n);
>> colormap(map)
>> subplot(2,1,1)
>> [xx,yy] = meshgrid(0:n,[0 1]);
>> c = [1:n+1;1:n+1];
>> pcolor(xx,yy,c)
>> set(gca,'Yticklabel','')
>> title('Figure 28.1a: Pcolor of Copper')
>> subplot(2,1,2)
>> rgbplot(map)
>> xlim([0,n])
>> title('Figure 28.1b: RGBplot of Copper')
```

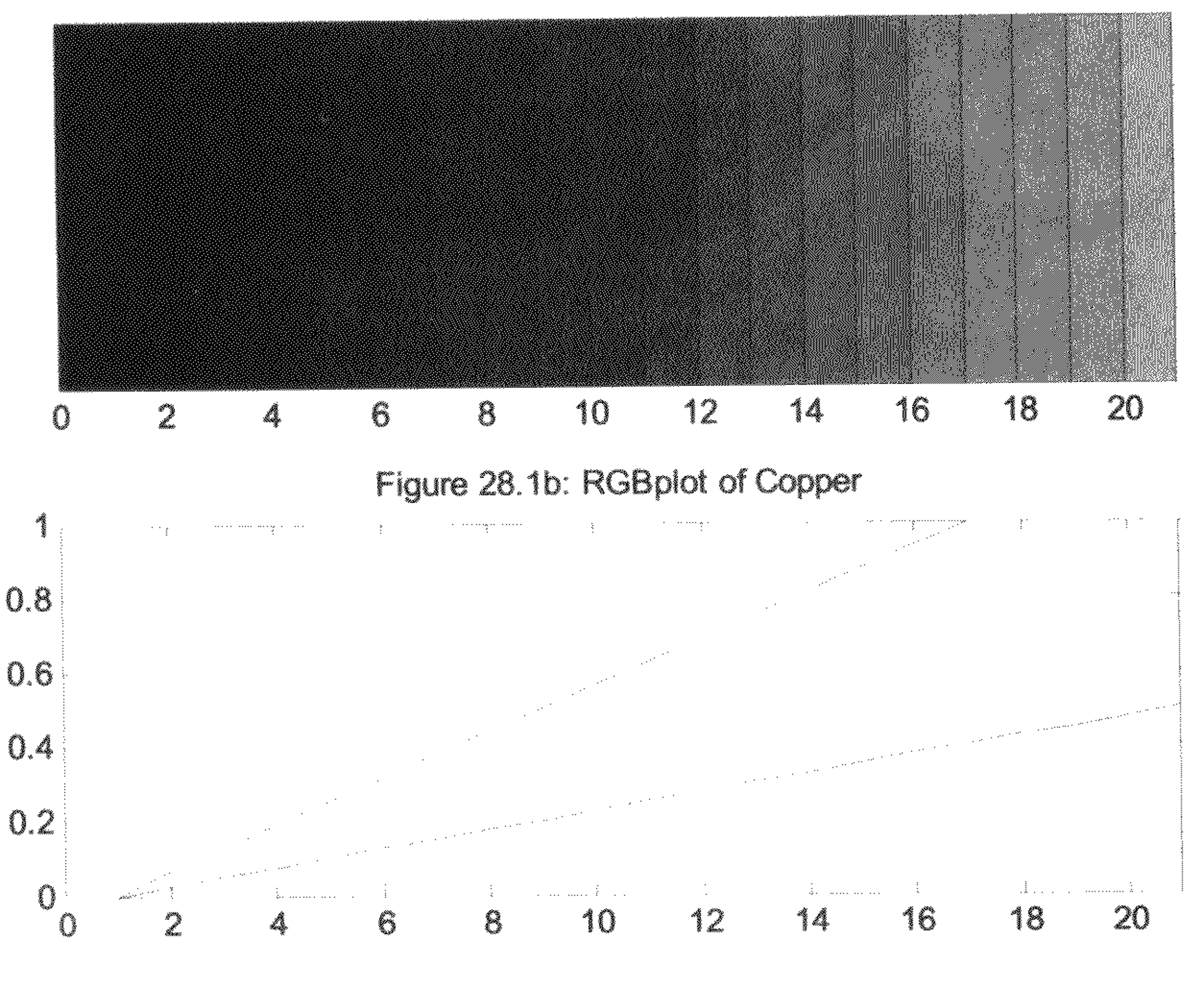


图 28.1　绘制铜色颜色表

图 28.1a 中显示了铜色颜色表的颜色变化情况，其中最左边的颜色带代表颜色表中的第一个颜色，最右边的颜色带代表颜色表中的最后一个颜色。而在图 28.1b 中，函数 rgbplot 仅仅画出了铜色颜色表中红色、绿色和蓝色分量的数值变化曲线，从而以可视化的方式分析了颜色表中各个颜色的三原色比重。

在用户绘制三维图形时，为了更清楚地说明所使用的颜色的信息，可以使用函数 colorbar 将图中的颜色信息在一个额外的辅助颜色条中显示，如下例所示：

```
>> mesh(peaks)
>> axis tight
>> colorbar
>> title('Figure 28.2: Colorbar Added')
```

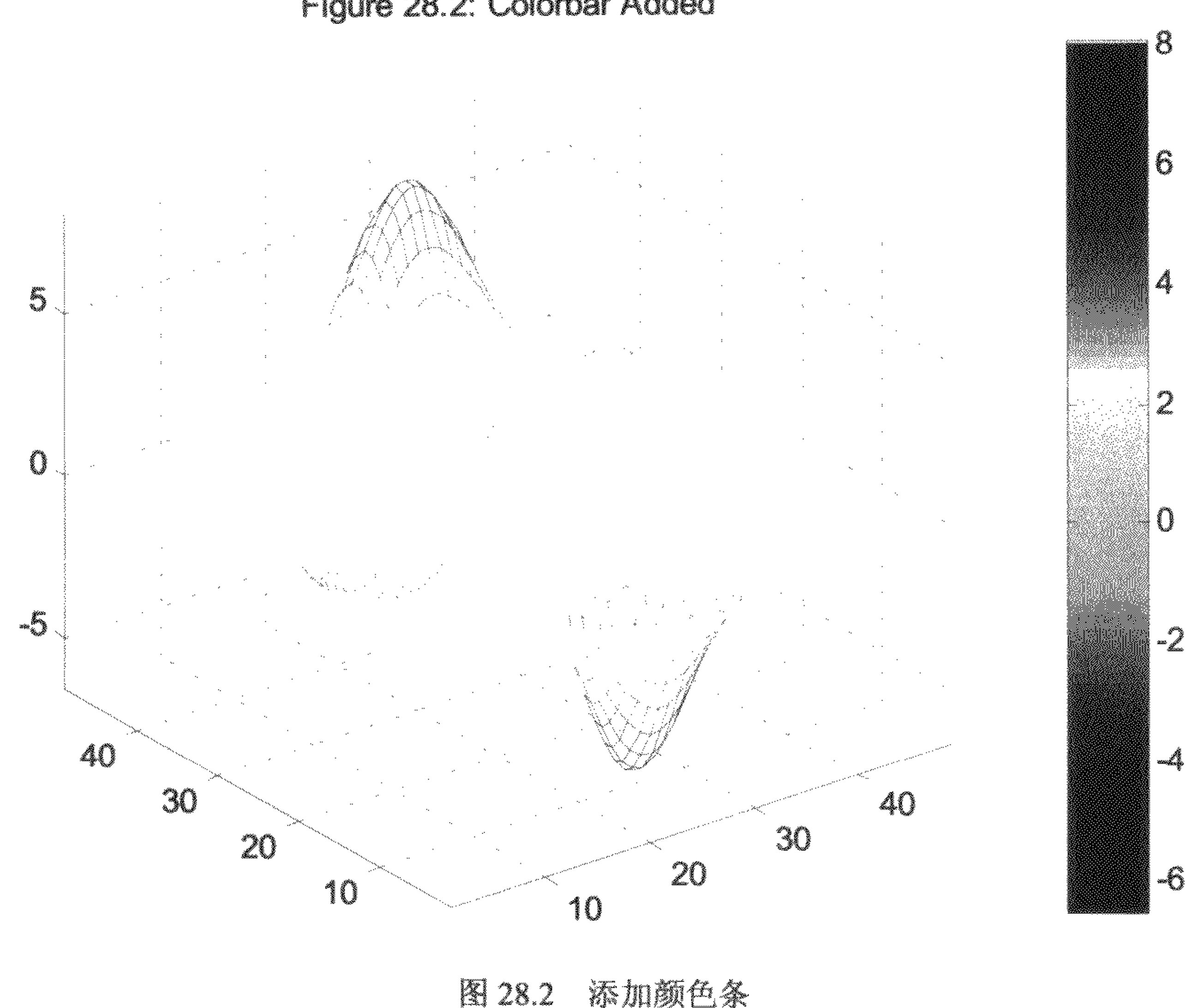


图 28.2 添加颜色条

由图上例中的曲面颜色与 z 坐标轴相关，因此，额外的辅助颜色条与 z 轴平行，并且颜色条中颜色的变化与 z 轴中 z 值的变化一致。

28.4 颜色表的创建和修改

本章第一节讲到，颜色表是用数组的形式表示的，因此，用户可以利用数组处理方式处理颜色表。例如，brighten 函数就是采用数组处理方式来调整一个给定颜色表中各个颜色的强度。brighten(beta)命令会根据 beta 的大小将当前的颜色表进行亮化（当 0＜beta≤1 时）或者暗化（当-1≤beta＜0 时）处理。当用户执行 brighten(beta)命令后，再执行 brighten(-beta)命令就可以将颜色表恢复到原来的状态。如果将 brighten(beta)的执行结果赋给一个变量，如 newmap=brighten(beta)，则将会创建一个经过亮化或暗化后的颜色表 newmap，原来的颜色表保持不变。用户也可以针对一个指定的颜色表进行亮化或暗化处理，如命令 mymap=brighten(cmap,beta)将对颜色表 cmap 进行修正，并将修正后的颜色表返回到 mymap，而 cmap 颜色表不会改变。

用户也可以先生成一个 m×3 的数组 mymap，再利用 colormap(mymap)命令将该数组转化为一个颜色表。需要注意的是，mymap 的列数必须等于 3，并且每个元素值均介于 0 和 1 之间，否则，colormap 函数将会报告出错信息。

在默认情况下，Matlab 总是将颜色表理解成 RGB 值，读者有时候需要使用 HSV 值表示颜色。鉴于此，Matlab 提供了两个函数 rgb2hsv 和 hsv2rgb 实现红-绿-蓝（RGB）颜色标准和色度-饱和度-亮度（HSV）颜色标准之间的转换。

用户可以通过对颜色表进行线性组合创建一个新的颜色表，但要保证所得结果满足颜色表维数（必须是 3 列）和元素值（必须介于 0 和 1 之间）的限制。例如，前面表格中的 pink 颜色表其实是由下式得到的：

```
pinkmap = sqrt(2/3*gray + 1/3*hot);
```

在正常的情况下，Matlab 将使用一个颜色表中所有的颜色来绘制用户的所有数据，也就是说，Matlab 将用颜色表中的第一个颜色绘制用户数据中最小的值，用颜色表中的最后一个颜色绘制用户数据中的最大值。如果不想像上述方式那样使用颜色表，可以使用函数 caxis 改变颜色表的使用方式，例如可以将整个颜色表用于用户数据的一个子集，或者将当前颜色表中的某一部分应用到整个用户数据集上。

命令[cmin,cmax]=caxis 将返回颜色表中第一个和最后一个颜色所对应的最小和最大数据值。在正常的情况下，这些数值通常被设置为用户数据的最小值和最大值。例如，在用户执行 mesh(peaks)命令时，将生成一个由 peaks 函数声明的网格图，并将 caxis 设置为[-6.5466,8.0752]，分别代表 z 坐标轴的最小和最大值。在这两个值之间的数据点所使用的颜色都是通过对颜色表中的颜色进行插值所得。

命令 caxis([cmin,cmas])将利用整个颜色表来表示位于 cmin 和 cmax 之间的数据。其中，大于 cmax 的数据将被绘制成与 cmax 相同的颜色，小于 cmin 的数据将被绘制成与 cmin 相同的颜色。在这种情况下，如果 cmin 小于用户数据的最小值，或者 cmax 大于用户数据的最大值，那么颜色表中与 cmin 或 cmax 对应的颜色将永远不会被用到，也就是说，只有与用户数据相对应的那部分颜色表中的颜色才会被用到。

另外，caxis('auto')或者其命令形式 caxis auto 用于将颜色表表示的数据范围恢复成默认值 cmin 和 cmax。

下面这个简单的例子显示了 caxis 函数的典型用法：

```
>> N = 17;
>> data = [1:N+1;1:N+1]';

>> subplot(1,3,1)
>> colormap(hsv(N))
>> pcolor(data)
>> set(gca,'XtickLabel','')
>> title('Figure 28.3: Auto Limits')
>> caxis auto      % automatic limits (default)

>> subplot(1,3,2)
>> pcolor(data)
>> axis off
>> title('Extended Limits')
>> caxis([-5,N+5]) % extend the color limits

>> subplot(1,3,3)
>> pcolor(data)
>> axis off
>> title('Restricted Limits')
>> caxis([5,N-5]) % restrict the color limits
```

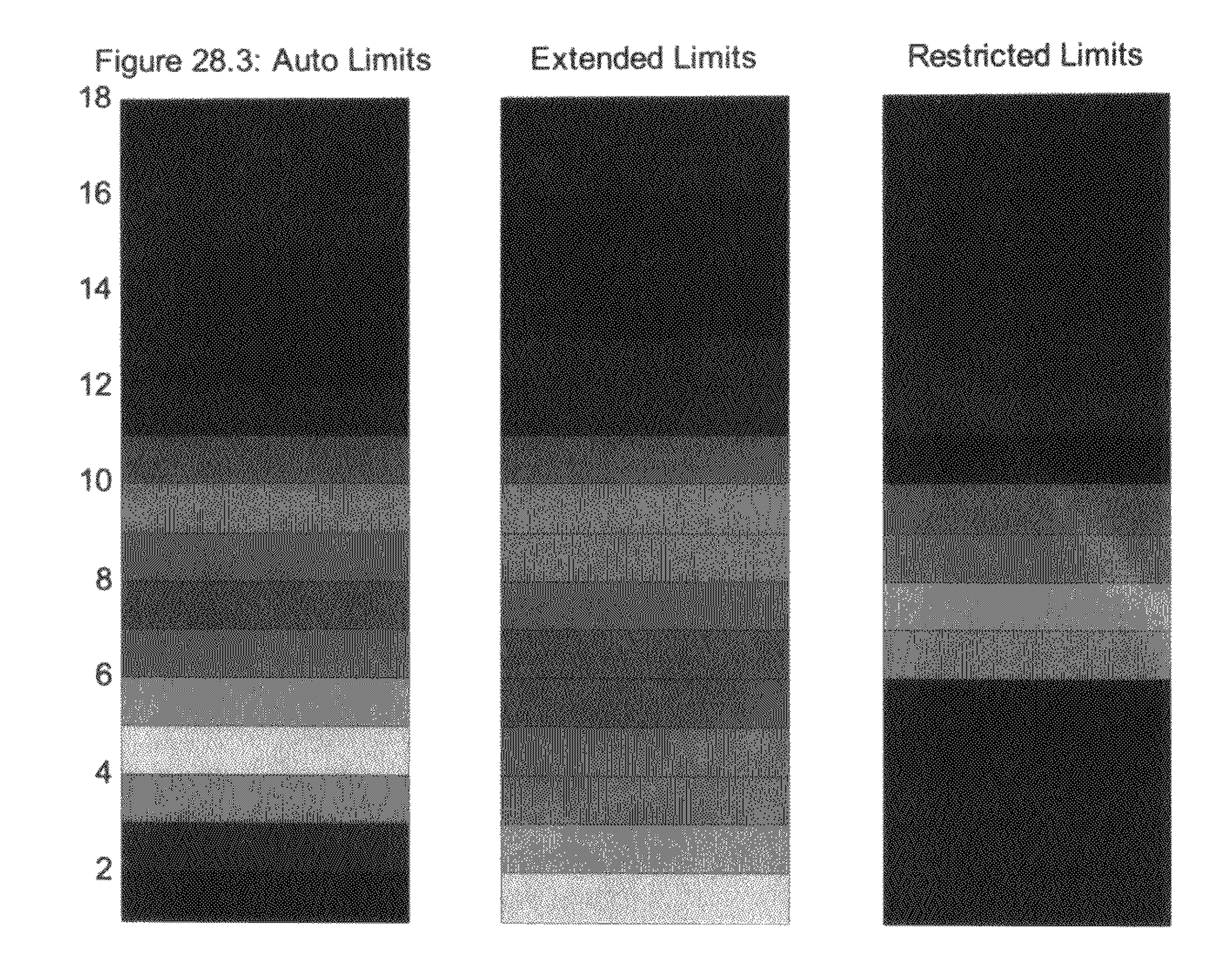


图 28.3　颜色表的取值范围

在图 28.3 中并排的 3 个图形中，最左边的图形是颜色表的默认使用方法，它将整个颜色表用于所有的用户数据；中间的图形只使用部分颜色表来绘制用户的数据，从效果上看，颜色表好像被放大了；最右边的图形则将整个颜色表应用到了部分用户数据中，从效果上看，颜色表只占数据显示的一部分，好像被缩小了，另外，所有超出颜色表表示范围的数据都用颜色表两端的颜色进行绘制。

28.5　用颜色描述第四维

从上一章我们可以看到，在默认情况下，大部分曲面绘制函数（如 mesh 和 surf）都只根据 z 轴的值来改变图形显示的颜色，除非用户指定了一个颜色参数。也就是说，命令 surf(X,Y,Z)与命令 surf(X,Y,Z,Z)是等价的。仅仅将颜色应用于 z 轴虽然会得到一幅色彩斑斓的图形，但却无法提供更多的额外信息。为了更好地利用颜色，用户通常希望将颜色用来描述 3 个坐标轴没有反映出的数据属性（如数组的第四维）。为了做到这一点，需要将一个指定的数组作为三维绘图函数的颜色参数（即第四个数据参数）传递给它。

如果绘图函数的颜色参数是一个向量或者矩阵，那么 Matlab 会将该参数换算成颜色表的索引值。该参数可以是任何与其他 3 个参数同维的实值向量或矩阵。下面给出了一个使用颜色参数的例子：

```
>> x = -7.5:.5:7.5;              % data
>> [X Y] = meshgrid(x);          % create plaid data
>> R = sqrt(X.^2 + Y.^2)+eps;    % create sombrero
```

```
>> Z = sin(R)./R;

>> subplot(2,2,1)
>> surf(X,Y,Z,Z)          % default color order
>> colormap(gray)
>> shading interp
>> axis tight off
>> title('Figure 28.4a: Default,Z')

>> subplot(2,2,2)
>> surf(X,Y,Z,Y)          % Y axis color order
>> shading interp
>> axis tight off
>> title('Figure 28.4b: Y axis')

>> subplot(2,2,3)
>> surf(X,Y,Z,X-Y)  % diagonal color order
>> shading interp
>> axis tight off
>> title('Figure 28.4c: X - Y')

>> subplot(2,2,4)
>> surf(X,Y,Z,R)          % radius color order
>> shading interp
>> axis tight off
>> title('Figure 28.4d: Radius')
```

Figure 28.4a: Default,Z

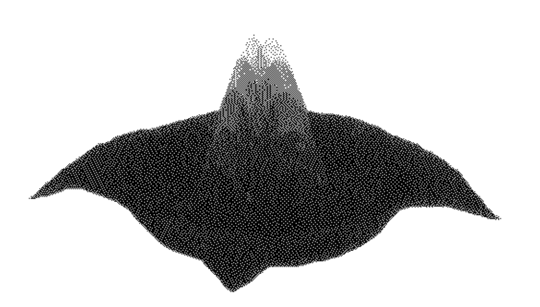


Figure 28.4b: Y axis

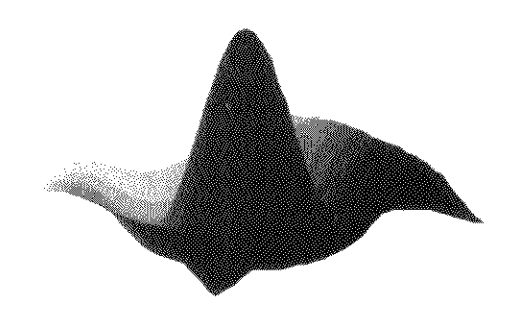


Figure 28.4c: X - Y

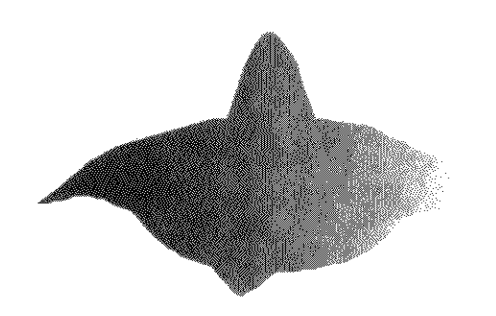


Figure 28.4d: Radius

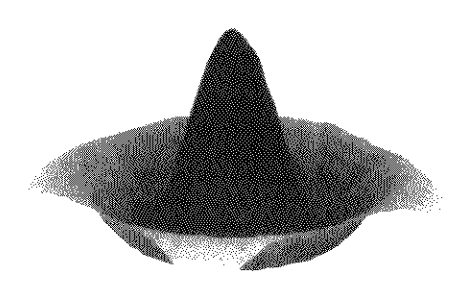


图 28.4　使用颜色绘制第四维

图 28.4 演示了利用颜色参数表示数组第四维的 4 个简单方法。当有第四个参数传递给绘图函数时，Matlab 会根据该参数对颜色表进行插值操作，以便使第四维的颜色显示具有连续性。第四个参数可以是前 3 个参数的一个函数，也可以是一个完全独立的变量。

利用函数 del2 和 gradient，用户还可以用颜色来分别表示曲率和斜率，如下例所示：

```
>> subplot(2,2,1)
>> surf(X,Y,Z,abs(del2(Z)))          % absolute Laplacian
>> colormap(gray)
>> shading interp
>> axis tight off
>> title('Figure 28.5a: |Curvature|')

>> subplot(2,2,2)
>> [dZdx,dZdy] = gradient(Z);   % compute gradient of surface
>> surf(X,Y,Z,abs(dZdx))         % absolute slope in x-direction
>> shading interp
>> axis tight off
>> title('Figure 28.5b: |dZ/dx|')

>> subplot(2,2,3)
>> surf(X,Y,Z,abs(dZdy))         % absolute slope in y-direction
>> shading interp
>> axis tight off
>> title('Figure 28.5c: |dZ/dy|')

>> subplot(2,2,4)
>> dR = sqrt(dZdx.^2 + dZdy.^2);
>> surf(X,Y,Z,abs(dR))           % absolute slope in radius
>> shading interp
>> axis tight off
>> title('Figure 28.5d: |dR|')
```

Figure 28.5a: |Curvature|

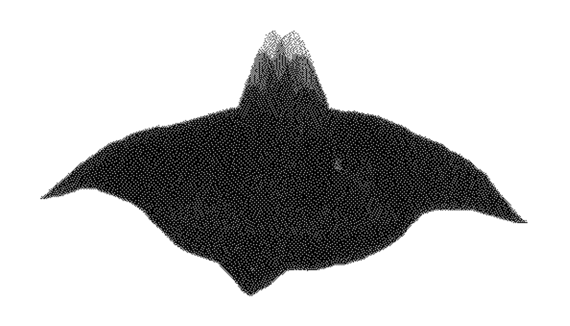


Figure 28.5b: |dZ/dx|

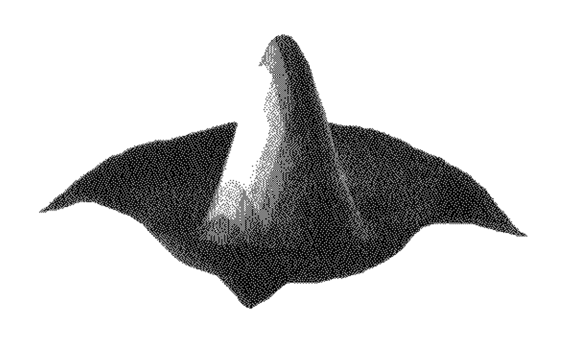


Figure 28.5c: |dZ/dy|

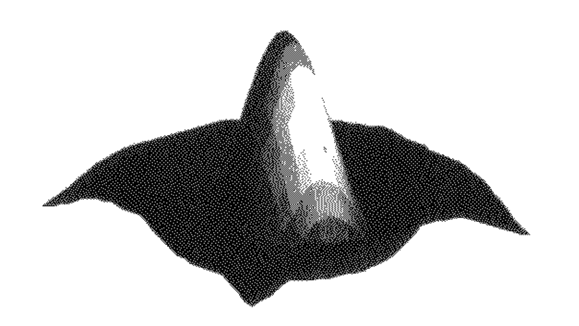


Figure 28.5d: |dR|

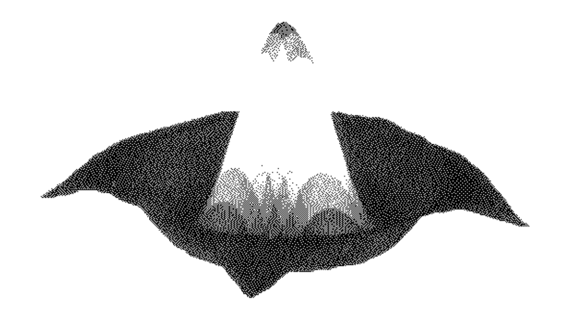


图 28.5 使用颜色表示斜率和曲率

上例中，函数 del2 是离散拉普拉斯函数，它根据表面的曲率来应用颜色。函数 gradient 则对表面上两个坐标轴方向的梯度或斜率进行近似。从上面的例子可知，利用颜色的确可以为绘制好的表面提供另外一维。

28.6 光照模型

在本节之前讨论的图形函数（如 pcolor、fill、fill3、mesh 和 surf 等）所绘制的图形都使用了默认的光照效果：即利用从各个方向照射过来的发散光线进行照明。这种光照方法有利于更好地在“图形”窗口中展示绘制对象的特点，并增强用户可视化分析数据的能力。尽管默认的光照方式可以非常清晰地展示数据，但本节仍将向读者展示不同的光照模型，以增强图形显示的实际效果。

函数 shading 是形成阴影的一个函数，它有 3 种阴影方式：小平面（faceted）阴影，平面（flat）阴影和插值（interpolated）阴影（它们都在上一章进行了阐述）。虽然 shading 函数在形成阴影时需要耗费大量的计算时间，但它却可以提高所显示图形的视觉效果。

用户可以添加一个或者多个光源来仿真一个物体的光照部位与阴影部位。在 Matlab 中，光源生成函数为 light 函数，该函数生成一个沿矢量[1 0 1]方向，从无穷远处照来的白光光源。生成光源后，用户可以使用 lighting 函数从以下 4 个不同的光照模型中为物体选择一种光照效果：none（忽略任何光源），flat（默认模型），phong 以及 gouraud。上述每一个模型都使用不同的算法来改变物体的外观：flat 光照模型在物体的每个面使用统一的颜色；gouraud 光照模型根据顶点的颜色对表面颜色进行插值；phong 光照模型对每个表面的顶点的法线进行插值，并计算每个像素处的反光。正如颜色表是“图形”窗口的属性一样，光照是坐标轴的一个属性。因此，“图形”窗口中的每个坐标轴都可以单独地设置光照。下面的例子验证了 4 种光照模型的效果：

```
>> subplot(2,2,1)
>> sphere
>> light
>> shading interp
>> axis square off
>> lighting none
>> title('Figure 28.6a: No Lighting')

>> subplot(2,2,2)
>> sphere
>> light
>> shading interp
>> axis square off
>> lighting flat
>> title('Figure 28.6b: Flat Lighting')

>> subplot(2,2,3)
>> sphere
>> light
>> shading interp
>> axis square off
```

```
>> lighting gouraud
>> title('Figure 28.6c: Gouraud Lighting')

>> subplot(2,2,4)
>> sphere
>> light
>> shading interp
>> axis square off
>> lighting phong
>> title('Figure 28.6d: Phong Lighting')
```

Figure 28.6a: No Lighting

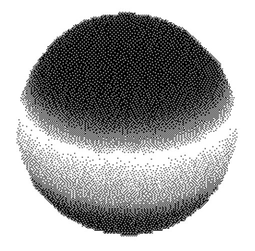


Figure 28.6b: Flat Lighting

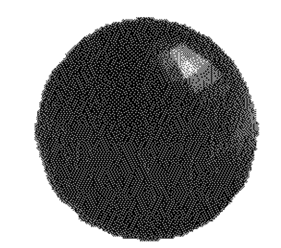


Figure 28.6c: Gouraud Lighting

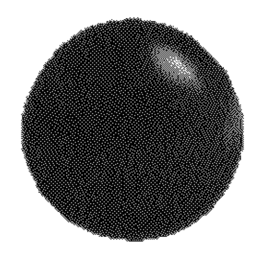


Figure 28.6d: Phong Lighting

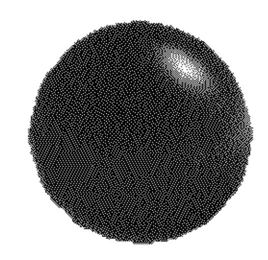


图 28.6　使用光照模型

除了光照之外，物体沿某一坐标轴的外观可以通过调整其表面上的外部反射特性（或称反射系数）来改变。反射系数由以下元素构成：

（1）环境光线——图中均匀漫反射光线的强度。
（2）散射反射——弱漫反射光线强度。
（3）镜面反射——强漫反射光线强度。
（4）镜面指数——控制镜面“热点”的尺寸或范围。
（5）镜面颜色反射——确定表面颜色对反射的影响大小。

用户可以利用函数 material 获得一些预先定义的表面反射属性。该函数的可选参数包括：shiny、dull、metal 和 default（用来存储默认的表面反射属性）。另外，material([ka kd ks n sc])这样的函数调用格式（其中 n 和 sc 都是可选的），用于设置物体沿某一坐标轴方向的环境光线强度、散射反射强度、镜面反射强度、镜面指数，以及镜面颜色反射等元素。下面给出了一个物体反射特性的例子：

```
>> subplot(2,2,1)
>> sphere
```

```
>> colormap(gray)
>> light
>> shading interp
>> axis square off
>> material default
>> title('Figure 28.7a: Default Material')

>> subplot(2,2,2)
>> sphere
>> light
>> shading interp
>> axis square off
>> material shiny
>> title('Figure 28.7b: Shiny Material')

>> subplot(2,2,3)
>> sphere
>> light
>> shading interp
>> axis square off
>> material dull
>> title('Figure 28.7c: Dull Material')

>> subplot(2,2,4)
>> sphere
>> light
>> shading interp
>> axis square off
>> material metal
>> title('Figure 28.7d: Metal Material')
```

Figure 28.7a: Default Material

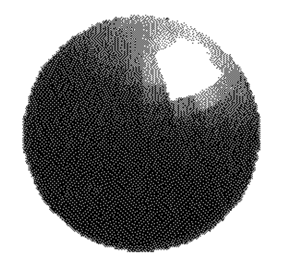


Figure 28.7b: Shiny Material

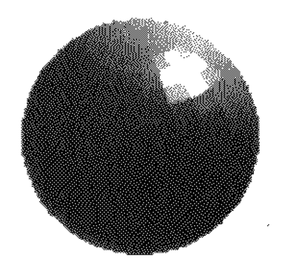


Figure 28.7c: Dull Material

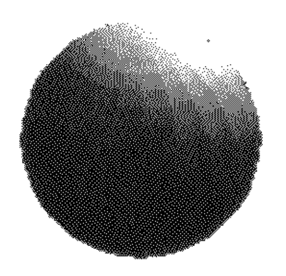


Figure 28.7d: Metal Material

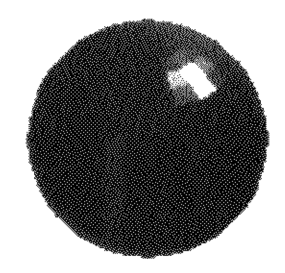


图 28.7 使用表面反射属性

上面的例子仅仅使用了 light 函数十分有限的功能，它生成了一个位于无穷远处向指定的位置发射的白光。实际上，light 是一个句柄图形对象生成函数（句柄图形函数将在第 30 章中讨论），提供了用户可以设置的多个不同属性，包括光线的颜色、位置以及类型等。其中，类型指光线既可以是一个指定位置的点光源，也可以是沿着某条射线的无穷远处的光源。下面给出了一条设置光源属性的语句：

```
>> Hl = light('Position',[x,y,z],'Color',[r,g,b],'Style','local');
```

该语句生成了一个位于(x,y,z)处的光源，光的颜色为[r,g,b]，并且声明了光源的类型是一个点（'local'）而不是一个指向无穷远的射线（'infinite'）。另外，上述语句还将光源对象句柄（H1）保存起来，便于用户在以后的使用过程中改变该光源的属性，如：

```
>> set(Hl,'Position',[1 0 1],'Color',[1 1 1],'Style','infinite');
```

该语句将用句柄 H1 定义的光源恢复成它原来的默认设置。（关于句柄图形的详细信息，请参见本书第 31 章。）

28.7　小结

总为小结，我们将与颜色和光照有关的 Matlab 函数列在了下面的表格中：

函数	描述
light	光照对象生成函数
lighting	设置光照模式（flat,gouraud,phong 或者 none）
lightangle	球坐标中的位置光照对象
material	设置反射的物质类型（default,shiny,dull 或者 metal）
camlight	设置与摄像头相关的光照对象
brighten	亮化或暗化颜色表
caxis	设置或获取颜色轴的限制
diffuse	找出表面散射反射
specular	找出表面镜面反射
surfnorm	计算表面法线
colorbar	生成颜色条
colordef	定义默认颜色属性
colormap	设置或获取图形窗口颜色表
colormapeditor	用于创建颜色映射表的图形用户接口（GUI）函数
hsv2rgb	将色度-饱和度-亮度颜色值转换为红-绿-蓝模式值
rgb2hsv	将红-绿-蓝颜色值转换为色度-饱和度-亮度模式值
rgbplot	绘制颜色表
shading	阴影模式（flat,faceted 或者 interp）
spinmap	使颜色旋转
whitebg	将图形窗口设置成白色背景
graymon	将图形窗口设置成灰度默认值，以便在单色显示器中显示

（续表）

函数	描述
autumn	带红色和黄色阴影的颜色表
bone	带有蓝色的灰度颜色表
cool	带有天蓝色和粉色阴影的颜色表
copper	线性铜色调颜色表
flag	带交替的红、白、蓝和黑色的颜色表
gray	线性灰度颜色表
hot	黑、红、黄、白基色颜色表
hsv	色度、饱和度、亮度（HSV）颜色表
jet	HSV 颜色表的变种（开始是蓝色，结尾是红色）
lines	基于线颜色的颜色表
prism	带交替的红、橙、黄、绿、蓝和紫色的颜色表
spring	带洋红和黄色阴影的颜色表
summer	带绿色和黄色阴影的颜色表
winter	带蓝色和绿色阴影的颜色表

Chapter 29

图像、视频和声音

Matlab 对多媒体数据的处理能力也很强大。首先，Matlab 提供了一系列命令和函数用于显示和处理图像。在 Matlab 中，图像数据通常被创建或保存为标准的双精度浮点数，有时也可以创建或者保存为 8 位或 16 位的无符号整数。Matlab 能够读写多种格式的图像文件，也可以用 load 和 save 命令来将图像数据保存在 MAT 文件中。另外，Matlab 还提供了创建和播放视频动画的命令。如果用户的系统支持声音，Matlab 也提供了一些声音函数用于对声音文件进行处理。

29.1 图像

在 Matlab 中，一幅图像通常由一个图像数据矩阵构成，有时候可能还需要一个与之相对应的颜色表矩阵。Matlab 的图像数据矩阵共有 3 种类型，即：索引图像数据矩阵、亮度图像数据矩阵和真彩图像数据矩阵（也称为 RGB 图像数据矩阵）。

索引图像是带有颜色表矩阵的，图像数据矩阵中的数据通常被解释成指向颜色表矩阵的索引号。图像颜色表矩阵可以是上一章讲到的任何有效的颜色表：即任何包含了有效 RGB 数据的 m×3 的数组。如果索引图像的图像数据数组为 X(i,j)，颜色表数组为 cmap，则每个图像像素 $P_{i,j}$ 的颜色就是 cmap(X(i,j),:)。这要求 X 中的数据值必须是位于[1 length(cmap)]范围之内的整数。如果用户已经获得图像数据和颜色表，可以使用下面的命令显示这幅图像：

```
>> image(X); colormap(cmap)
```

亮度图像的图像数据通常表示该图像的亮度值。该类型图像通常用于显示由灰度或单色颜色表染色的图像，有时也用于其他颜色表染色的图像。亮度图像对数据范围没有要求，不一定要像索引图像那样位于[1 length(cmap)]范围之内。用户可以指定亮度图像的数据范围，并且将其作为指向颜色表的索引。如下面的例子：

```
>> imagesc(X,[0 1]); colormap(gray)
```

将 X 的值限制在[0 1]之间，并将 0 指向颜色表的第一个颜色，将 1 指向颜色表的最后一个颜色，介于 0 和 1 之间的数据被用来作为指向颜色表中其他颜色的索引。如果在上面

的语句中省略[0 1]，则意味着不对 X 进行限定，也就是说，X 的数据范围是[min(min(X)) max(max(X))]。

真彩色（也叫 RGB）图像通常由一个包含有效 RGB 值的 m×n×3 的数组创建。该数组的行和列声明了像素的位置，页则声明了图像中每一个像素的颜色值。也就是说，像素 $P_{i,j}$ 将用 X(i,j,:)所声明的颜色绘制。由于真彩色图像已经将颜色信息包含在图像数据中，因此它不需要颜色表。如果计算机硬件不支持真彩色图像（例如，它只有一块 8 位显卡），那么 Matlab 就利用颜色近似和抖动来显示图像。真彩色图像的显示比较简单，如下所示：

```
>> image(X)
```

其中，X 是一个 m×n×3 的真彩色图像。X 可以包含双精度数据，也可以包含 unit8、unit16 类型的数据。

如果图像在默认的坐标轴上显示，则通常会由于坐标轴宽高比与图像高宽比不匹配导致图像的扭曲变形。为解决这一问题，可输入如下命令：

```
>> axis image off
```

该命令对坐标轴属性进行了设置，使得坐标轴高宽比与图像相匹配，并在显示图像时将坐标轴和坐标标签隐藏。有时用户希望在显示图像时把每个图像数据都显示出来，也就是使图像中的每个数据都对应屏幕上的一个像素点。这时，用户需要对 figure 和 axes 属性进行设置，如下所示：

```
>> load clown                    % sample image
>> [r,c] = size(X);              % pixel dimensions
>> figure('Units','Pixels','Position',[100 100 c r])
>> image(X)
>> set(gca,'Position',[0 0 1 1])
>> colormap(map)
```

这里，通过将 figure 的宽度和高度设置成等于图像的宽和高，figure 被设置显示出和图像完全相同的像素数量。然后，axes 位置被设定为以标准化单位占据整个的 figure。

除了上边使用的 clown.mat 之外，Matlab 安装程序中还有一些示例图像。Matlab 的 demos 子目录下有 cape.mat、clown.mat、detail.mat、durer.mat、flujet.mat、gatlin.mat、mandrill.mat 和 spine.mat。这些图像中的每一个都可以通过输入下面的指令来显示：

```
>> load filename
>> image(X), colormap(map)
>> title(caption)
>> axis image off
```

29.2 图像格式

在 Matlab 中，默认的数值型数据类型为 double。这指的是双精度、64 位浮点数。Matlab 对其他数据格式，比如图像的 16 位字符数据类型（unit16）和 8 位无符号整型（unit8），提供了有限的支持。

命令 image 和 imagesc 可以显示 8 位和 16 位图像，而不用预先把它们转换成 double 型。但是，unit8 数据值的范围是[0 255]，这是在标准图形文件格式中支持的数据格式，unit16 的数据值的范围是[0 65535]。

对于被索引的索引图像，image 通过自动地提供相应的偏移量将值 0 映射到有 256 个条目的颜色表的第一个条目，而将值 255 映射到最后的一个条目。因为被索引图像的 double 数据的正常范围为[1 length(cmap)]，因此在 unit8 和 double 或者 unit16 和 double 之间的转换需要将数据值平移 1。另外，对 unit8 数组的数学运算还没有定义。因此，为了对无符号整型数执行数学运算，它们必须被转换成 double 格式，例如：

```
>> Xdouble = double(Xuint8) + 1;
>> Xuint8 = uint8(Xdouble - 1);
```

这些命令将 Xunit8 中的 unit8 数据转换成 double，然后再转换回来。对于 8 位亮度和 RGB 图像而言，数据值的范围通常是[0 255]，而不是[0 1]。为了显示 8 位强度和 RGB 图像，需要使用下面的命令：

```
>> imagesc(Xuint8,[0 255]);colormap(cmap)
>> image(Xuint8)
```

向 double 的转换也可以进行标准化，例如：

```
>> Xdouble = double(Xuint8)/255;
>> Xuint8 = uint8(round(Xdouble*255));
```

RGB 图像中的 8 位颜色数据在显示的时候被自动地标定。例如，在使用双精度数的时候，白色通常是[1 1 1]。如果相同的颜色储存在 8 位数据中，白色就被表示为[255 255 255]。

Matlab 中提供的可选图像处理工具箱包含了很多图像处理函数。如果用户需要经常处理图像，这个工具箱是很有用处的。

29.3 图像文件

可以用多种不同的文件格式来将图像数据储存在文件中和重新载入到 Matlab 中。通常的 Matlab 函数 save 和 load 支持 double、unit8 或者 unit16 格式的图像数据，就像这些函数支持其他 Matlab 变量和数据类型一样。当保存非标准颜色表的被索引图像或者强度图像的时候，一定要保存颜色表及所显示的图像数据，例如：

```
>> save myimage.mat X map
```

Matlab 还利用函数 imread 和 imwrite 支持多种工业标准的图像文件格式。可以用函数 imfinfo 获得关于图形文件内容的信息。Imread 的帮助文档，给出了关于图像读取格式和特性的广泛信息，下面显示了其中的一部分。

```
Supported file types
--------------------
JPEG  Any baseline JPEG image; JPEG images with some
      commonly used extensions; 8-bit and 12-bit lossy
      compressed RGB and grayscale images; 8-bit and 12-bit
```

```
        lossless compressed RGB images; 8-bit, 12-bit, and
        16-bit lossless compressed grayscale images

   TIFF Any baseline TIFF image, including 1-bit, 8-bit, and
        24-bit uncompressed images; 1-bit, 8-bit, and 24-bit
        images with packbits compression; 1-bit images with
        CCITT compression; 16-bit grayscale, 16-bit indexed,
        and 48-bit RGB images; 24-bit and 48-bit ICCLAB and
        CIELAB images; 32-bit and 64-bit CMYK images; and 8-bit
        tiled TIFF images with any compression and colorspace
        combination listed above.

   GIF  Any 1-bit to 8-bit GIF image

   BMP  1-bit, 4-bit, 8-bit, 16-bit, 24-bit, and 32-bit uncompressed
        images; 4-bit and 8-bit run-length encoded (RLE) images

   PNG  Any PNG image, including 1-bit, 2-bit, 4-bit, 8-bit,
        and 16-bit grayscale images; 8-bit and 16-bit
        indexed images; 24-bit and 48-bit RGB images

   HDF  8-bit raster image datasets, with or without an
        associated colormap; 24-bit raster image datasets

   PCX  1-bit, 8-bit, and 24-bit images

   XWD  1-bit and 8-bit ZPixmaps; XYBitmaps; 1-bit XYPixmaps

   ICO  1-bit, 4-bit, and 8-bit uncompressed images

   CUR  1-bit, 4-bit, and 8-bit uncompressed images

   RAS  Any RAS image, including 1-bit bitmap, 8-bit indexed,
        24-bit truecolor and 32-bit truecolor with alpha.

   PBM  Any 1-bit PBM image.  Raw (binary) or ASCII (plain) encoded.

   PGM  Any standard PGM image.  ASCII (plain) encoded with
        arbitrary color depth.  Raw (binary) encoded with up
        to 16 bits per gray value.

   PPM  Any standard PPM image.  ASCII (plain) encoded with
        arbitrary color depth. Raw (binary) encoded with up
        to 16 bits per color component.
```

调用 imwrite 的语法根据图像的类型和文件格式的变化而变化。imwrite 的帮助文档给出了关于图像保存格式和特性的广泛信息，下面显示了其中的一部分。

```
   Table: summary of supported image types
   ---------------------------------------
   BMP  1-bit, 8-bit and 24-bit uncompressed images

   TIFF Baseline TIFF images, including 1-bit, 8-bit, 16-bit,
        and 24-bit uncompressed images; 1-bit, 8-bit, 16-bit,
        and 24-bit images with packbits compression; 1-bit
        images with CCITT 1D, Group 3, and Group 4 compression;
        CIELAB, ICCLAB, and CMYK images
```

```
JPEG Baseline JPEG images

PNG  1-bit, 2-bit, 4-bit, 8-bit, and 16-bit grayscale
     images; 8-bit and 16-bit grayscale images with alpha
     channels; 1-bit, 2-bit, 4-bit, and 8-bit indexed
     images; 24-bit and 48-bit truecolor images; 24-bit
     and 48-bit truecolor images with alpha channels

HDF  8-bit raster image datasets, with or without associated
     colormap; 24-bit raster image datasets; uncompressed or
     with RLE or JPEG compression

PCX  8-bit images

XWD  8-bit ZPixmaps

RAS  Any RAS image, including 1-bit bitmap, 8-bit indexed,
     24-bit truecolor and 32-bit truecolor with alpha.

PBM  Any 1-bit PBM image, ASCII (plain) or raw (binary) encoding.

PGM  Any standard PGM image. ASCII (plain) encoded with
     arbitrary color depth. Raw (binary) encoded with up
     to 16 bits per gray value.

PPM  Any standard PPM image. ASCII (plain) encoded with
     arbitrary color depth. Raw (binary) encoded with up
     to 16 bits per color component.

PNM  Any of PPM/PGM/PBM (see above) chosen automatically.
```

29.4 影片

Matlab 中的动画采用了两种形式。一种形式是，如果生成一个图像序列所需要的计算足够快，那么就可以设置 figure 和 axes 属性，使得屏幕绘制以足够快的速度进行，这样动画从视觉上看起来就是平稳的。另一种形式是，如果计算需要大量的时间，或者结果得到的图像过于复杂，用户就必须生成一个影片。

在 Matlab 中，函数 getframe 和 movie 提供了捕获和演示影片所需要的工具。函数 getframe 对当前的图像进行一次快照，movie 在这些快照都被捕获之后，回头重新播放这些帧序列。函数 getframe 的输出是一个结构，它包含了 movie 所需要的所有信息。捕获多个帧的过程仅仅就是向这个结构里边添加元素的过程。请看下边这个例子：

```
% moviemaking example: rotate a 3-D surface plot
[X,Y,Z]=peaks(50);          % create data
surfl(X,Y,Z)                % plot surface with lighting
axis([-3 3 -3 3 -10 10]     % fix axes so that scaling does not change
axis vis3d off              % fix axes for 3D and turn off axes ticks etc.
shading interp              % make it pretty with interpolated shading
colormap(copper)            % choose a good colormap for lighting
for i=1:15                  % rotate and capture each frame
   view(-37.5+15*(i-1),30)  % change the viewpoint for this frame
```

```
    m(i)=getframe;          % add this figure to the frame structure
end
cla         % clear axis for movie
movie(m)    % play the movie
```

通过不断地旋转顶峰的表面并且每旋转一次就捕获一个帧，上边这个脚本文件生成了一个影片。最后，在清除 axes 之后，播放了这个影片。变量 m 包含了一个结构数组，其每个数组元素包含了一个帧，例如：

```
>> m
m =
1x15 struct array with fields:
    cdata
    colormap

>> size(m(1).cdata)
ans =
    342     306     3
```

保存了图像 cdata 的颜色数据是一个真彩色或者 RGB 位图图像数据。因此，axes 内容的复杂性并没有影响存储一个影片所需要的字节数。用像素表示的 axes 的大小决定了图像的大小，因此也就决定了储存一个影片所需要的字节数量。

29.5 图像工具

可以用函数 im2frame 和 frame2im 实现被索引图像和影片之间的转换。例如：

```
>> [X,cmap] = frame2im(M(n))
```

这条命令将影片矩阵 M 的第 n 帧转换成一个被索引图像 X 和相关的颜色表 cmap。类似地：

```
>> M(n) = im2frame(X,cmap)
```

这条命令将被索引图像 X 和颜色表 cmap 转换成影片矩阵 M 的第 n 帧。请注意，im2frame 可以用来将一系列图像转换成一个影片，其方式和 getframe 将一系列 figure 或者 axes 转换成一个影片的方式相同。

29.6 声音

Matlab 除了提供高层处理函数 audiorecorder 和 audioplayer 以外，还提供了许多底层函数来处理声音。例如，函数 sound(y,f,b)将向量 y 中的信号以采样频率 f 发送到计算机的扬声器中。变量 y 中超出了[-1 1]范围之外的值被省略。如果 f 被省略，就使用默认的采样频率 8192Hz。如果有可能，Matlab 用 b 位/秒播放这个声音。大多数的平台都支持 b=8 或者 b=16。如果 b 被省略，就使用 b=16。

函数 soundsc 和 sound 基本相同，只是其向量 y 中的值都被标定在范围[-1 1]之内，而不是把超出这个范围的值省略。这使得声音可以尽可能的大，而不用将过大的声音省略。

还可以用一个额外的参数来使用户将 y 值的某个范围和整个声音范围对应起来。其格式为 soundsc(y,…,[smin smax])。如果这个参数被省略了，默认的范围就是[min(y) max(y)]。

在 Matlab 中，还提供了两个工业标准的声音文件格式。Matlab 可以对 NeXT/Sun 音频格式（file.au）文件和 Microsoft WAVE 格式（file.wav）文件进行读写操作。

NeXT/Sun 音频声音存储格式支持 8 位 μ 律压缩、8 位线性和 16 位线性格式的多通道数据。auwrite 的最常用调用格式是 auwrite(y,f,n, 'method', 'filename')，其中 y 是采样数据，f 是用赫兹表示的采样频率，b 表示在编码器中的位数，'method'是一个表示编码方法的字符串，'filename'是一个表示输出文件名的字符串。y 的每一列代表了一个不同的通道。y 中任何超出了范围[-1 1]的值在写入文件之前就被忽略了。参数 f，n 和'method'都是可选的。如果这些参数被省略了，那么就默认地将这些参数设置为 f=8000，n=8，'method'='mu'。参数'method'必须是'linear'或者'mu'。如果文件名字符串没有包含后缀，Matlab 就自动给它加上'.au'的后缀。

μ 律压缩和线性格式之间的转换可以利用函数 mu2lin 和 lin2mu 来进行。关于这两个函数所涉及到的确切的转换过程的信息请参见联机帮助文档。

多通道 8 位或者 16 位 WAVE 声音存储格式声音文件可以用 wavwrite 函数来生成。其最常用的调用格式为 wavwrite(y,f,n, 'filename')，其中 y 是采样数据，f 是以赫兹为单位的采样频率，b 声明了在编码器中的位数，'filename'是一个声明了输出文件的字符串。y 的每一列都代表了一个单独的通道。y 中任何超出了范围[-1 1]的值在写入文件之前都被忽略了。参数 f 和 n 是可选的。如果这些参数被省略了，Matlab 就使用其默认值 f=8000 和 n=16。如果文件名字符串没有带后缀名，那么 Matlab 就会自动给它加上'.wav'的后缀。

auread 和 wavread 都有相同的调用语法和选项。最常用的调用格式为[y,f,b]=auread('filename',n)，这条语句载入由字符串'filename'声明的声音文件，并将采样数据返回给 y。如果这个'filename'字符串没有给出后缀（.au 或者.wav），那么 Matlab 就将相应的后缀名添加到文件名后边。y 中的数据值都在范围[-1 1]之内。如果需要输出如上所示的 3 个输出参数，那么就在 f 和 b 中分别返回以赫兹为单位的采样频率和每个采样的位数。如果给出了参数 n，那么就只返回文件中每个通道的前 n 个采样。如果 n=[n1 n2]，那么只返回每个通道从第 n1 个到第 n2 个之间的采样。形如[samples,channels]=wavread('filename', 'size')的调用格式将返回文件中音频数据的数量大小，而不是数据本身。这种调用格式对于预先分配存储空间或者估计资源使用量来说是很有用处的。

29.7　小结

下表对 Matlab 中所提供的图像、影片和声音功能进行了总结。

函数	描述
image	创建索引或真彩色（RGB）图像对象
imagesc	创建亮度图像对象
colormap	将颜色表应用到图像
axis image	调整坐标轴刻度使其适应图像

（续表）

函数	描述
unit8	将变量类型转换为无符号 8 位整型
uint16	将变量类型转换为无符号 16 位整型
double	将变量类型转换为双精度数
imread	读取图像文件
imwrite	写图像文件
imfinfo	获取图像文件信息
getframe	将影片帧放在结构体中
movie	从影片结构体中播放影片
frame2im	将影片帧转换成图像
im2frame	将图像转换成影片帧
avifile	生成 avi 影片文件
addframe	将影片帧添加到 avi 影片文件中
close	关闭 avi 影片文件
aviread	读取 avi 影片文件
aviinfo	获取 avi 影片文件的信息
movie2avi	将 Matlab 格式的影片转换为 avi 格式
audiorecorder	声音录制对象
audioplayer	声音播放对象
audiodevinfo	获取声音设备信息
sound	将向量以声音的形式播放
soundsc	将向量进行归一化处理并以声音的形式播放
wavplay	播放 WAVE 格式的声音文件
wavrecord	利用 Windows 的音频输入设备记录声音
wavread	读取 WAVE 格式的声音文件
wavwrite	写 WAVE 格式的声音文件
auread	读取 NeXT/SUN 格式的声音文件
auwrite	写 NeXT/SUN 格式的声音文件
lin2mu	将线性音频转换为 μ 律压缩音频
mu2lin	将 μ 律压缩音频转换为线性音频

Chapter 30

打印和导出图形

Matlab 图形是进行数据可视化和数据分析的强有力工具。除了在屏幕上观察和分析图形外，用户也经常希望生成图形的硬拷贝输出（即打印输出）或将图形应用在其他应用程序中。为此，Matlab 提供了一个灵活实用的系统来打印图形和将图形生成多种不同格式的输出，如 EPS 和 TIFF 格式等，这些输出格式可以被大多数应用程序导入并使用。

有一点需要读者注意，在打印或者导出图形时，图形将被重新绘制。也就是说，用户在屏幕上看见的图形与用户在打印纸上或在导出文件中所看到的图形是不一样的。在默认情况下，Matlab 将选择不同的绘制器（如 painter，zbuffer 或 OpenGL 等），来改变各坐标的标记符号分布，以及打印和导出的图形大小。实际上，Matlab 尽量使用户在打印和导出图形时达到“所见即所得（WYSIWYG）”的目标，但这是 Matlab 提供的附加功能，并不是其默认功能。

图形的打印和导出涉及到打印机驱动程序、打印机协议、图形文件类型、绘制器、位图和向量图描述格式、dpi（每英寸像素点）选择、彩色和黑白图形、图形压缩格式、用户使用平台等因素，因此，读者在打印和导出图形时，需要综合考虑这些因素中的一项或多项的组合。如果用户采用缺省模式打印和导出图形，可以不考虑这些因素。但如果用户要打印出具有特定属性的图形，或要将图形导出以供字处理文档使用，就必须要考虑上面的因素，因此，用户需要花费大量的时间来调整输出图形，直到它满足希望的属性。

本章主要介绍 Matlab 提供的图形打印和导出功能。与其他一些功能模块一样，Matlab 也提供了图形打印和导出的菜单栏选项和“命令”窗口函数。其中菜单栏方法使用起来比较方便，但灵活性稍差；窗口函数法灵活性很强，但需要用户掌握较多的基础知识。

> 用户可以在命令窗口中输入命令>>doc print 查看 print 函数的在线帮助文档，该文档提供了关于图形打印和导出的详细信息。

30.1 利用菜单打印和导出图形

当用户使用图形窗口显示图形时，会发现该窗口顶部包含一个菜单栏（有时还可能包含一个或多个工具栏），该菜单栏上有诸如 File、Edit、View、Insert、Tools、Desktop、Window

和 Help 等一系列菜单。在这些菜单中，File 菜单中包含了打印和导出当前图形的菜单项。另外，Windows 系统的 Edit 菜单中也包含了将当前图形导出到系统剪贴板的菜单项。打开 File 菜单，可以发现该菜单下包含了 Export Setup、Page Setup、Print Setup、Print Preview 和 Print 等菜单项，点击每一项都会弹出一个对话框用于设置不同的打印和导出属性。下面对这些菜单项进行逐一描述。

Export Setup 菜单项用于将当前图形保存为多种图形格式。Export Setup 对话框中提供了一个 Export…对话框，当用户激活这个对话框后，可以为要保存的图形选择存储路径、文件名和文件类型。另外，用户也可以在 Export Setup 对话框中设置要保存的图形的大小和特性。

如果用户选择 Page Setup 菜单项（或者在命令窗口中输入 pagesetupdlg 命令），就可以打开一个带有多个表项菜单的对话框，该对话框用于设置当前图形在打印和导出时进行重绘的一些特性。另外，该对话框也是指定打印和导出图形的大小、方向、布局以及其他特性的主要界面。

在 Windows 操作系统中，Print Setup 菜单项用于打开一个标准的对话框（也可以在命令窗口中输入 print –dsetup 命令打开这个对话框），该对话框用于选择打印时的打印机，以及设置纸张大小、纸张来源和页面方向等选项。如果用户的 Matlab 安装在一个 UNIX 系统中，则 Print Setup 菜单项（或 printdlg –setup 命令）将打开一个和 UNIX 的 Print 对话框类似的对话框，惟一的差别在于两者的对话框标题不同。在 UNIX 中，点击 Print 对话框中的 OK 按钮可以关闭这个对话框，但不把图形发送到打印机缓存（或打印机文件）中。UNIX 的 Print Setup 对话框除了包含打印机选择、要打印的文件名、打印机驱动名、打印的图形大小、打印的坐标轴和标签等众多选项，还包含一个 Option 按钮，点击该按钮，用户可以在弹出的对话框中设置其他的一些附加选项，这些选项包括选择绘图器、设置打印输出的分辨率等。无论用户的 Matlab 安装在哪个操作系统，一旦设置了 Print Setup 中的打印选项，这些设置将在整个 Matlab 执行期间保持有效，并且在打印时忽略 Page Setup 对话框中的选项设置。

Print Preview 菜单项（或 printpreview 命令）用于打开一个窗口显示图形打印的效果。该窗口中还提供了几个按钮用来随时调用 Page Setup 对话框对图形打印进行设置上的改动，另外，也可以直接调用 Print 对话框来打印图形。

Print 菜单项（或 printdlg 命令）用于打开一个标准的 Print 对话框。在这个对话框中，用户可以选择使用的打印机和打印的份数。另外，用户也可以在这个对话框中选择将图形打印到一个文件中。如果用户使用的是 UNIX 系统，可以在该对话框中选择打印机、文件名、打印机驱动程序、以及固定打印坐标轴和刻度线等选项。其中也提供了一个 Options 按钮，单击该按钮将弹出一个对话框，允许用户设置一些附加选项，这些附加选项在 Page Setup 对话框中也可以找到。

> Page Setup 选项只应用于当前图形，并和当前图形一起被保存起来。Print Setup 选项则在整个 Matlab 执行周期保持不变，并用其中的设置将 Page Setup 中的相关选项设置覆盖。Print 选项只能应用于当前图形，并将当前图形的 Page Setup 或 Print Setup 中的相关选项设置覆盖。

在 Windows 操作系统中，Edit 菜单提供了利用系统剪贴板进行文件导出的功能。单击 Edit 菜单下的 Copy Figure 菜单项，用户可以根据选项对话框中的选项设置（选项对话框可以通过选择 Edit 菜单下的 Copy Option 菜单项打开）将当前图形放入剪贴板。在拷贝当前图形时，既可以将其拷贝成位图（BMP）格式，也可以拷贝成增强型图元文件（EMF）格式。用户也可以使用一个模板来改变拷贝图形的线条宽度和字体大小，以及其他一些默认选项。注意，上述选项设置只在将图形拷贝到剪贴板时有效，在图形打印和导出时是无效的。

30.2 利用命令行打印和导出图形

在命令窗口中，只需要一个 print 命令就可以处理所有的打印和导出任务。print 命令提供了多个选项参数，用来指定不同的打印和导出操作。print 命令的一般调用语法为：

```
>> print -device -option -option filename
```

上面所有的参数都是可选参数，既可以有，也可以没有。其中，-device 用来指定要使用的设备驱动程序，-option 用来指定一个或者多个打印或导出选项，如果用户要将图形导出到一个文件，而不是直接将其送到打印机端口，可以使用 filename 参数指定导出的文件名。由于命令与函数具有二元性，因此 print 也可以当作函数进行调用。例如，上面的命令格式可以等效为下面的函数格式：

```
>> print('-device','-option','-option','filename')
```

通常，print 命令中的各选项之间的先后顺序是不受限制的，它们可以按任何顺序进行指定。下表给出了 print 命令的各选项的字符串名称及其描述（'filename'除外）：

选项名称	描述
-adobecset	在打印或导出时选择 PostScript 默认字符集编码（只适用于早期的 PostScript 打印驱动程序和 EPS 文件格式）
-append	在打印或导出时将图形附加在现存的文件后边（只适用于 PostScript 打印驱动程序）
-cmyk	在打印或导出时使用 CMYK 颜色而非 RGB 颜色（只适用于 PostScript 打印驱动程序和 EPS 文件格式）
-device	指定使用的打印驱动程序
-dsetup	显示 Print Setup 对话框（只适用于 Windows 系统）
-fhandle	指定要打印或者导出的图形的数值句柄
-loose	在打印或导出时使用 loose 类型的 PostScript 边框（只适用于 PostScript，EPS 和 GhostScript 格式）
-noui	在打印或导出时不显示 uicontrol 对象
-opengl	使用 OpenGL 算法进行绘图（即绘制成位图格式）
-painters	使用 Painter 算法进行绘图（即绘制成向量格式）
-Pprinter	指定要使用的打印机的名字（只适用于 UNIX 系统）

（续表）

选项名称	描述
-rnumber	指定打印或导出时的分辨率（单位为每英寸像素数，即 dpi）。在大多数打印设备中均可设置该参数；另外，除了 EMF 和 ILL 文件之外，Matlab 的大部分内置函数也可设置该参数；大部分 GhostScript 导出格式不可设置该参数。在默认情况下，Z 缓冲和 OpenGL 绘图算法的分辨率为 150dpi，Painter 绘图算法的分辨率为 864dpi
-swindowtitle	指定用来打印和导出的 SIMULINK 系统窗口的名字
-v	显示 Print 对话框（只适用于 Windows 系统）
-zbuffer	用 Z 缓冲算法绘图（即绘制成位图格式）

30.3 打印机和导出文件格式

print 函数可以将图形打印或导出到多种输出设备（包括打印机和多种文件类型）中。其中大部分打印机都是通过 GhostScript 打印机驱动程序得到了 Matlab 的支持，该驱动程序可以将 PostScript 打印机代码转换成本地的打印机代码。这个转换过程对用户而言是透明的，但打印时所用的字体必须是 PostScript 支持的字体。

函数 print 还支持以多种图形格式将图形导出到文件或剪贴板。下表列出了 Matlab 7 中 print 命令支持的图形文件格式，用户可以在 print 命令的-device 选项中设置这些文件格式：

图形文件格式	描述
-dbmp16m	24 位位图格式（只适用于 Windows 系统，需要用到 GhostScript 驱动）
-dbmp256	带固定颜色表的 8 位位图格式（只适用于 Windows 系统，需要用到 GhostScript 驱动）
-dbmp	24 位位图格式（只适用于 Windows 系统）
-dmeta -deps	EMF 格式（只适用于 Windows 系统） EPS 第一级，黑白图形，包括灰度图形
-depsc	EPS 第一级，彩色图形
-deps2	EPS 第二级，黑白图形，包括灰度图形
-depsc2	EPS 第二级，彩色图形
-hdf	24 位 HDF 格式（只适用于 Windows 系统）
-dill	Adobe 图形格式
-djpeg -djpeg*NN*	压缩质量为 75 的 24 位 JPEG 格式（用 Z 缓冲算法绘制） 压缩质量设为 *NN* 的 24 位 JPEG 格式
-dpbm	PBM 普通格式（只适用于 UNIX 系统，需要用到 GhostScript 驱动）
-dpbmraw	PBM 原始格式（只适用于 UNIX 系统，需要用到 GhostScript 驱动）
-dpcxmono	1 位 PCX 格式（只适用于 Windows 系统，需要用到 GhostScript 驱动）
-dpcx24b	24 位彩色 PCX 格式（只适用于 Windows 系统，需要用到 GhostScript 驱动）
-dpcx256	8 位彩色 PCX 格式（只适用于 Windows 系统，需要用到 GhostScript 驱动）
-dpcx16	16 位彩色 PCX 格式（只适用于 Windows 系统，需要用到 GhostScript 驱动）

（续表）

图形文件格式	描述
-dpcx	8 位彩色 PCX 格式（只适用于 Windows 系统）
-dpgm	PGM（可移植灰度映射）普通格式　（只适用于 UNIX 系统，需要用到 GhostScript 驱动）
-dpgmraw	PGM（可移植灰度映射）原始格式　（只适用于 UNIX 系统，需要用到 GhostScript 驱动）
-dpng	24 位 PNG 格式
-dppm	PPM（可移植像素映射）普通格式　（只适用于 UNIX 系统，需要用到 GhostScript 驱动）
-dppmraw	PPM（可移植像素映射）原始格式　（只适用于 UNIX 系统，需要用到 GhostScript 驱动）
-dtiff	TIFF 格式（使用 Z 缓冲算法绘制）
-tiff	利用 TIFF 格式预览 EPS 格式的图形，必须同时使用 EPS 设备规范选项

除了 print 外，Matlab 还提供了一些函数用于将图形导出到图像文件中。例如，函数 getframe、inwrite、avifile 和 addframe 就可以将当前图形创建和保存成图像文件，详见第 29 章。

30.4　PostScript 支持

所有的 PostScript 设备以及那些使用 GhostScript 驱动程序的设备都只能提供有限的字体支持，这些设备通常包括打印设备和保存图像的设备。下面的表格显示了 PostScript 支持的字体以及对应的 Windows 的标准字体。注意：表中没有列出的字体表示被映射为 Windows 标准 Courier 字体，另外，由于 Windows 设备-dwin 和-dwinc 都使用标准的 Windows 打印驱动程序，因此能够支持所有的字体。

PostScript 字体	Windows 中的等价字体
AvantGarde	
Bookman	
Courier	Courier New
Helvetica	Arial
Helvetica-Narrow	
NewCenturySchlBk	New Century Schoolbook
Palatino	
Symbol	
Times-Roman	Times New Roman
ZapfChancery	
ZapfDingbats	

如果用户的打印机支持 PostScript，那么在打印机工作时就需要用到一个内置的 PostScript 驱动程序。该驱动程序有两个级别的版本规范：第一级 PostScript 是一个老版本规范，有个别打印机可能需要这个版本；第二级 PostScript 提供了比第一级版本规范更精简、更快速的代码，因此如果用户的打印机支持该版本，要尽可能地使用这一级规范。

如果用户的打印机是一台彩色打印机，那么最好使用一个彩色打印机的驱动程序。黑白打印机或灰度打印机既可以使用单色的打印机驱动程序，也可以使用彩色打印机驱动程序。不过，当用户用彩色打印驱动程序驱动一个黑白打印机时，会出现颜色抖动现象，从而使得有些线条和文本不那么清晰。如果用户使用黑白打印驱动程序来打印彩色线条时，这些线条将被打印成黑色。如果用户的打印机是一个黑白打印机，并且使用彩色打印驱动程序，当用户打印一个彩色线条时，该线条将会被打印成灰色。此时，如果线条宽度不是足够宽，打印的效果和预览的效果将不会有太大的差别。

在 Matlab 中，PostScript 也支持具有插值阴影的表面和碎片。在打印时，相应的 PostScript 文件将包含表面或碎片顶点处的颜色信息，这些颜色信息会要求打印机在打印时进行阴影插值。由于每个打印机特性不同，阴影插值过程可能会占用大量的时间，有时还会导致打印机错误。解决这一问题的一个方法是增加网孔表面的细微程度，并使用平面投影。另一种确保打印输出与屏幕显示图像匹配的方法是使用足够高分辨率的 Z 缓冲算法或 OpenGL 算法来绘制图形并打印输出。此时，将输出一个很大的位图格式的图形文件，但不需要打印机进行插值计算，从而可以节省打印时间，并减少出错概率。

30.5 选择绘制器

绘图器用于将图形数据（包括顶点数组和颜色数据）转换成适合显示、打印或导出的图形格式。这些图形格式主要包括两种：位图（或光栅）图形格式和向量图形格式。

位图图形格式中包含了栅格中各点的颜色信息。很明显，对于一幅固定尺寸的图像，栅格分得越细，栅格中的点数越多，图形的分辨率就越高，文件大小也越大。增加栅格中每个点的颜色位数会使图形中可用的颜色总数增大，同时也会增加图形文件的大小。不过，增加图形内容的复杂度不会对图形文件的大小产生影响。

向量图形格式中包含了通过创建点、线和其他几何对象重构原图形的指令。因此，向量图形可以很容易地改变大小，并且通常会得到比位图更高的分辨率。但是，随着图形中需要重构的对象数量增加，所需要的重构指令数也会增加，文件的大小将随之增大。甚至在有些点处，指令的复杂程度相当大，使得输出设备无法处理，最后造成图形无法在指定的输出设备上输出。

Matlab 7 支持 3 种绘制方法：OpenGL、Z 缓冲和 Painter's。其中 Painter's 使用向量格式绘制图形，OpenGL 和 Z 缓冲方法则用于生成位图。在默认情况下，Matlab 会根据图形特点和所使用的打印驱动程序或文件格式自动选择绘制器。

> 注意：Matlab 选择的用来进行打印或导出的绘制器不一定非要和用来进行屏幕显示的绘制器相同。

用户可以通过设置覆盖 Matlab 的默认选项。这样做可以使打印或导出的图形和屏幕显示效果一样，另外也可以避免将一个位图图形嵌入到一个向量格式的输出文件（例如 PostScript 或 EPS）中。

在有些情况下，必须或最好使用上述 3 种绘制器中的其中一种或两种，主要包括：

（1）如果图形使用了真彩色（RGB 颜色）而不是单色来表示图形的表面或碎片对象，那么该图形必须用一个位图绘制器以便准确地捕获颜色。

（2）HPGL（-dhpgl）和 Adobe 图形（-dill）格式的文件最好使用 Painter's 绘制器。

（3）JPEG（-djpeg）和 TIFF（-dtiff）格式的文件最好使用 Z 缓冲绘制器。

（4）光照效果不能使用向量格式的绘制器（如 Painter's）进行重建，必须使用位图绘制器。

（5）图形的透明属性只能利用 OpenGL 绘制器表现。

打印和导出时的绘制器既可以通过 print 对话框进行选择，也可以通过设置句柄图形属性进行选择。如果用户平台是 UNIX 系统，则可以点击 Print Setup 和 Print 对话框中的 Options 按钮，在打开的对话框中进行绘制器选择；如果用户平台是 Windows 或其他系统，则可以在 Page Setup 对话框中的 Axes and Figures 表项中进行绘制器选择。另外，用户也可以使用命令 print 中的-zbuffer、-opengl 和-painters 参数选项选择绘制器，此时，这些选项设置将覆盖用户在对话框中进行的相应选择。

30.6 句柄图形属性

某些句柄图形属性也会对图形的打印或导出产生影响。另外，打印和导出对话框中进行的一些设置也会改变当前图形的部分属性。下表列出了影响打印和导出的一些图形属性：

属性名称	**属性值选项列表，{ } 中为默认值**	**描述**
Color	[RGB vector]	设置图形的背景颜色
InvertHardcopy	[{on} | off]	确定是否将图形的背景颜色打印或导出。如果该属性设置为 on，不论 Color 属性是何值，都强制输出白色的图形背景。
PaperUnits	[{inches} | centimeters| normalized | points]	设置度量打印或导出图形的大小的单位
PaperOrientation	[{portrait} | landscape | rotated]	设置图形在纸张中打印的方向
PaperPosition	[left bottom width height] vector	设置图形在纸张或导出文件中的位置。其中 width 和 height 决定了打印或导出的图形大小

（续表）

属性名称	属性值选项列表，{ } 中为默认值	描述
PaperPositionMode	[auto | {manual}]	确定是否使用 PaperPosition 属性中的 width 和 height 值。若该属性设为 auto，就用图形窗口中所显示图形的宽度和高度来显示或导出该图形，即输出所见即所得（WYSIWYG）的图形
PaperSize	[width height] vector	以 PaperUnits 属性中设定的值为单位的纸张大小
PaperType	[{usletter} | uslegal | A0 | A1 | A2 | A3 | A4 | A5 | B0 | B1 | B2 | B3 | B4 | B5 | arch-A | arch-B | arch-C | arch-D | arch-E | A | B | C | D | E | tabloid | <custom>]	选择所使用的纸张类型。选定了纸张类型，也就设定了相应的纸张大小
Renderer	[{painters} | zbuffer | OpenGL]	设置使用的绘制器
RendererMode	[{auto} manual]	确定如何选择绘制器。如果该属性设置为 auto，Matlab 就自动选择绘制器，并独立显示、打印和导出图形。如果该属性设置为 manual，将使用 Renderer 属性设置的绘制器显示、打印和导出图形

坐标轴（axes）属性也会影响图形的打印和导出，尤其是坐标轴的刻度线模式。下面是图形打印和导出时的刻度线模式属性及其可选择的值：

```
XTickMode [{auto} | manual]
YTickMode [{auto} | manual]
ZtickMode [{auto} | manual]
```

在通常情况下，打印或导出的图形和显示器上的图形大小不同，因此，在默认条件（auto）下，Matlab 会对每个坐标轴上的刻度线的数字和位置重新标定。要使图形被打印或者导出时不改变坐标轴上的刻度线，则需要将上述属性设置为'manual'。

另外，用户在打印和导出时还可以使用下面给出的线条（line）属性使输出图形更加优化：

```
Color:[ 3-element RGB vector ]
LineStyle: [ {-} | - | : | -. | none ]
LineWidth: [ scalar ]
Marker: [ + | o | * | . | x | square | diamond | v | ^ | > | <
        | pentagram | hexagram | {none} ]
MarkerSize: [ scalar ]
MarkerEdgeColor: [ none | {auto} ] -or- a ColorSpec.
MarkerFaceColor: [ {none} | auto ] -or- a ColorSpec.
```

需要注意的是，在打印彩色线条时，打印结果与输出设备和打印驱动程序密切相关。如果用户使用的是彩色打印机并安装了彩色打印驱动程序，那么所打印出来的线条颜色通

常就是用户希望的颜色；如果用户安装了黑白打印驱动程序，那么打印结果就是黑白的线条；如果用户安装了彩色打印驱动程序，但却使用黑白打印机，那么所得到的线条就是灰度线条。由于灰度是通过色彩抖动产生的，因此可能会导致灰度线条无法与背景区分开来。另外，对于黑白打印图形，也会出现一些类似问题，例如当多条实线都被打印成黑色的时候，我们就无法区分原始的颜色信息。所以，在上述情况下，用户需要使用线条的'Color'、'LineStyle'、'LineWidth'和'Marker'等属性用来为绘制的图形添加一些其他可以加以区分的特征，以补充颜色信息的丢失。

最后，用户在打印或导出图形时还需要注意文本的属性，下面给出了用户可以使用的文本属性及其属性值：

```
Color: [ 3-element RGB vector ]
FontAngle: [{normal} | italic | oblique ]
FontName: [ font name ]
FontSize: [scalar]
FontUnits: [ inches | centimeters | normalized | {points} | pixels ]
FontWeight: [ light | {normal} | demi | bold ]
```

用户在打印或导出时，对字体的处理也是很必要的。例如，如果用户在打印或导出图形时，使图形尺寸变小，就需要增加标题和坐标轴标签处的文本字体的大小，以方便文本的阅读。如果用户在图形中使用的字体不是 Matlab 支持的用于 PostScript 打印输出的那 11 种字体，则用户最好在打印或导出前将字体变成这 11 种字体中的一种，这样可以避免出现意想不到的字体显示问题。另外，在打印输出中，粗体字的效果要比普通字体好。总之，我们在改变图形中的字体时，都是为了使打印或导出的图形更加美观。最后，当用户导出图形是为了用演示软件进行显示时，合理地利用字体属性是非常必要的，因为此时必须使图形中的字体足够大，并且能够和其他内容明显区分，以便观众即使在很远的距离也能清晰地看到并理解字体所表示的内容。

30.7　设置默认值

如果用户不对 Matlab 进行任何打印和导出的设置，则 Matlab 在打印和导出时将使用出厂的默认设置：

（1）打印的图形大小为 8×6 英寸（可能在美国以外发行的版本稍有不同）。
（2）打印方向为纵向。
（3）图形和坐标轴的背景颜色均为白色。
（4）纸张大小为 US 信纸（8.5×11 英寸）（可能在美国以外发行的版本稍有不同）。
（5）图形位于打印纸中央。
（6）图形会被剪裁以适应纸张大小。
（7）打印输出的图形是 RGB 图形（不是 CYMK）。
（8）会重新计算刻度线以适应绘图需要。
（9）绘制器由 Matlab 选择。
（10）使用 Uicontrol 方法打印。

（11）在 Unix 中打印设备为-dps2，在 Windows 平台中则为-dwin。

默认的打印设备是在$TOOLBOX/local/printopt.m 文件中设置的，用户可以直接打开并编辑这个文件来改变所使用的打印设备。如果用户没有对这个文件的写权限，可以用命令 edit printopt.m 在 Matlab 编辑窗口中编辑这个文件，然后将该文件另存到一个本地目录中，但要确保本地目录在 Matlab 路径列表中位于$TOOLBOX/local 之前。例如，如果用户希望在 UNIX 系统中使用彩色 PostScript 打印机打印图形，则可打开 printopt.m 文件，将代码 dev='-dpsc2'添加到指定的位置，然后保存这个文件即可。

除了打印设备外，其他选项都可以通过 startup.m 文件来设置。例如，set(0, 'DefaultFigurePaperType', 'A4')这条代码将纸张类型设为 A4。其中的句柄 0 表示在“根”这一级设置属性，'DefaultFigurePaperType'表示将后面的 A4 设置默认的'PaperType'属性值。用户在设置属性值时，在任何属性名前加上'Default'都表明将本次设置的值作为默认值使用。对于一个给定的对象，要将某一属性值设置为默认属性，必须在比该属性所在句柄图形层次高的级别进行。例如，默认的图形属性必须在“根”对象这一级进行设置，而默认的坐标轴属性必须在“图形”对象或者“根”对象这一级进行设置。总之，默认设置在“根”这一级进行是比较保险的方法。下面给出了几个设置默认属性的例子：

```
set(0,'DefaultFigurePaperOrientation','landscape');
set(0,'DefaultFigurePaperPosition',[0.25 0.25 10.5 8]);
set(0,'DefaultAxesXGrid','on','DefaultAxesYGrid','on');
set(0,'DefaultAxesLineWidth','1');
```

上述语句将图形和坐标轴的默认属性设置为：打印方向为风景（'landscape'）模式，图形位置为填满整个页面，显示图形中的坐标栅格（包括 x 轴和 y 轴），并且栅格的粗细为一个像素点。

除了上面介绍的内容以外，还有很多其他选项都可以利用句柄图形属性进行设置，并且大多数打印和导出属性都包含在图形（figure）和坐标轴（axes）属性之中。关于句柄图形属性的更多信息，请参见下一章。

30.8 发布

Matlab 7 新增加了发布功能，即允许用户使用不同的格式将用户自定义格式的导出结果（包括图形）发布出去。这样的话，用户在执行 Matlab 时，就可以用一个格式化的文档保存输出对象。这些格式化文档类型包括 HTML、XML 和 LaTeX，如果用户的系统是 Windows 系统，则用户可使用的格式化文档类型还包括 Microsoft Word 或 Powerpoint，以及 Notebook（记事本）等。本书第 36 章给出了一个使用记事本的例子。

要使 Matlab 实现发布功能，用户需要在相应的 M 文件中定义用于发布的代码单元。当一个 M 文件被发布时，该 M 文件执行后的所有输出（包括每个发布单元中的源代码、命令窗口的输出、图形窗口的输出）都将被转换成指定的格式，并通过指定格式的文档发布出去。Matlab 发布支持各种类型的文本标注，包括标题、副标题、描述性文本、公式以及 TeX 格式的符号等。这些文本中既可以包含粗体字、斜体字或预先定义格式的字体，也可以包含项目列表、等宽文本甚至是 HTML 链接。

用户可以通过选择 Matlab 编辑器中 File 菜单下的 Publish 选项进入发布窗口，也可以在命令窗口中直接键入 publish 命令进入。另外，用户可以在 Matlab Preferences 菜单中 Editor/Debugger 子项中设置发布选项。有关发布的一些特殊指令和示例请读者参考 Matlab 的联机帮助文档。

30.9 小结

Matlab 可以非常灵活地打印图形和创建多种图形格式的输出文件。这些输出文件大都可以被其他应用程序导入，但这些应用程序对这些图形文件的编辑能力都很有限。因此，如果用户希望获得最佳的输出效果，就需要在图形打印或导出之前进行编辑，或设置适当的选项。使用最广泛、最灵活的图形输出格式为 PostScript、EPS、EMF 和 TIFF，它们都是 Matlab 本身支持的格式，另外，Matlab 还可利用 GhostScript 将这些格式转换成其他格式。

Matlab 7 还支持各种格式的文档发布，发布的内容包括部分 M 文件代码、命令窗口输出、图形窗口输出、描述性文本以及注释等，发布的文档格式则包括 HTML、XML、LaTeX、Microsoft Word、Microsoft Powerpoint 等。

Chapter 31

句 柄 图 形

在 Matlab 中，句柄图形指一系列描述图形的表现形式和显示方式的底层图形特性函数的总称。通过交互式使用句柄图形对象及其属性，用户几乎可以对 Matlab 提供的图形特性进行任何方式的控制。随着 Matlab 版本的不断升级，许多句柄图形特性都可以通过 Matlab 的图形窗口中提供的各种菜单、上下文菜单、工具栏、控制面板、浏览器以及编辑器进行操作。借助这些交互式的工具，即使用户对句柄图形不甚了解，也可以轻松定制自己的图形特性。因此，只有当用户无法用这些交互式工具进行操作时，才有必要了解句柄图形函数的具体使用方法，并且，此时用户最好使用一个 M 文件来调用句柄图形函数。

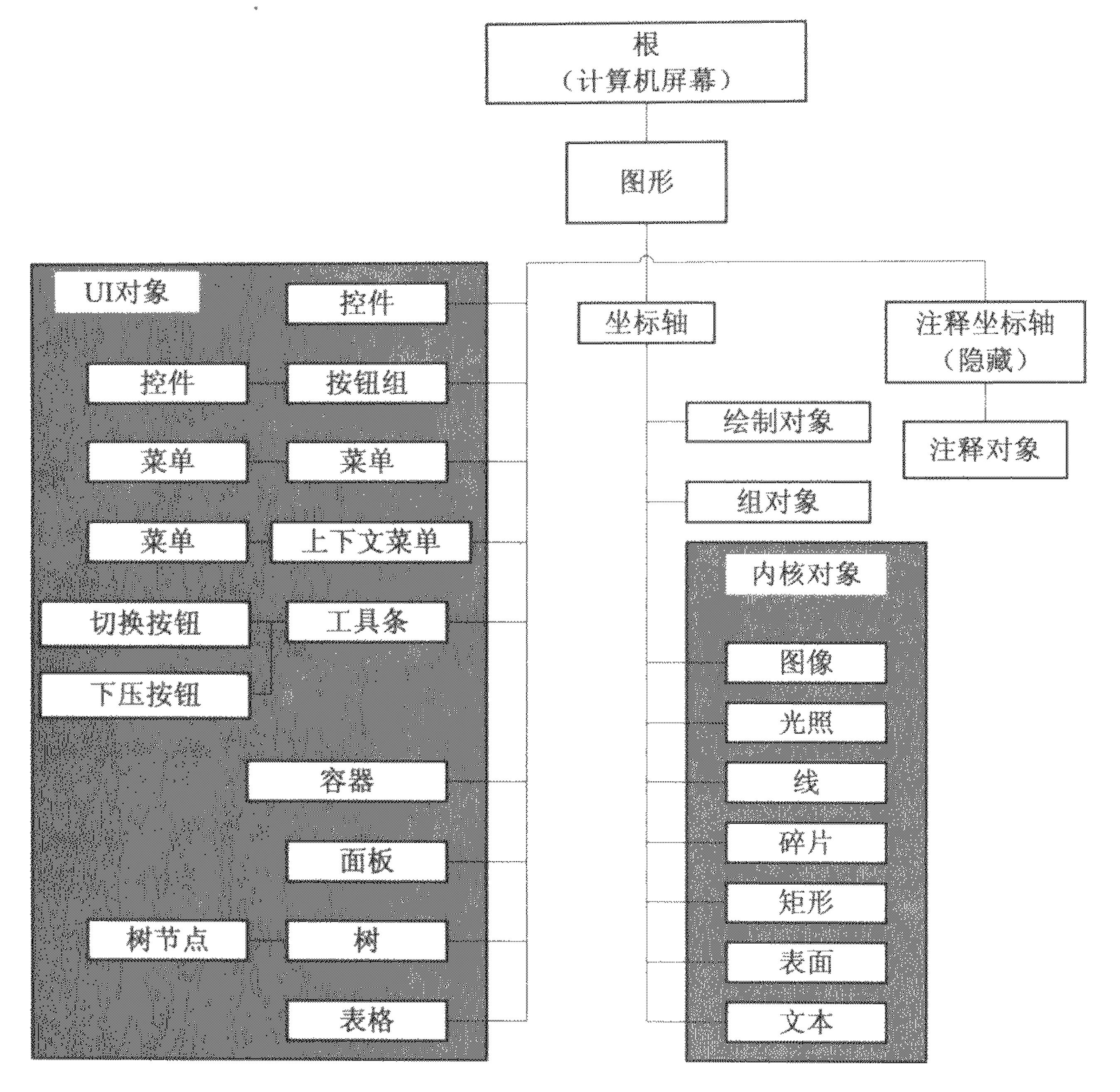
图形特性包含了多种多样、范围很宽的内容，为了说明它的用法，Matlab 几乎用了上千页的篇幅。因此，本文不可能也没有必要详尽地叙述句柄图形的每一项内容。

限于篇幅，本章将引导读者对句柄图形的特性有个大致了解，为读者日后更加熟练地使用复杂的句柄图形奠定基础。

31.1 对象

句柄图形的基本思想是：Matlab 的每一个可视部分都被视为一个对象，其中每个对象都有一个惟一的标识符（也称为句柄）与之对应，并且每个对象都包含用户可以修改的一组属性。这里的对象指构成一个统一整体的一组密切相关的数据和函数。在 Matlab 中，一个图形对象就是一个可以被单独处理的独立单元。

Matlab 中所有的绘图和图形函数都可以用来创建图形对象。只不过有些函数创建的是复合对象，有些函数创建的是内核对象。下图给出了 Matlab 中各图形对象之间的包含和隶属关系：



从上面的分级图可以看出，用户的计算机屏幕本身也是对象，称为根对象，它是所有其他对象的父对象。图形（figure）对象是根对象的子对象。坐标轴对象、注释坐标轴对象和各种用户接口（UI）对象则是图形对象的子对象。绘图（plot）对象、组（group）对象、和各种内核（core）对象是坐标轴对象的子对象。除了内核对象外，图形对象的其他子对象都称为复合对象。

上面的分级图是基于 Matlab 7 画出的，在 Matlab 6 中，该图会有所不同：首先 Matlab 6 中不存在隐藏的注释坐标轴对象、绘制对象和组对象；其次，虽然列出了内核对象和各种用户接口对象，但并没有用带底纹的方框将它们标示出来。Matlab 7 中的用户接口对象和内核对象与以前的 Matlab 版本完全相同，但绘制对象和组对象是 Matlab 7 新增加的对象。

31.2 对象句柄

在 Matlab 中，每个对象都有一个与之相对应的标识符，称为句柄，它实际上是一个双精度数。当一个对象被创建时，Matlab 就为该对象生成一个惟一的句柄。Matlab 约定：根对象（也就是计算机屏幕）的句柄为 0。命令 Hf_fig=figure 将生成一个新的图形，并将其句柄值返回给变量 Hf_fig。图形句柄值在正常情况下都是整数，并且会在图形窗口的标题栏中显示。其他对象的句柄值一般都是双精度浮点数。所有的对象生成函数都会返回它们所创建的对象的句柄值。

在 Matlab 7 之前的版本中，高级图形函数（参看第 26、27 章）在创建图形时将会返回一个列向量用于保存它创建的每个内核对象的句柄值。例如，命令 Hl=plot(…)将返回 plot 函数所创建的所有线对象的句柄值。另外，命令 Hs=surf(…)将返回一个表面对象的句柄。在 Matlab 7 中，许多高级图形创建函数将返回绘制对象的句柄。例如，命令 Hls=plot(…)将返回线列（lineseries）绘制对象的句柄。虽然 Matlab 7 的对象分级图有了一定的改变，但句柄图形的基本操作仍保持不变。因此，在阐述 Matlab 7 中的新增对象之前，我们先讨论一下句柄图形对象的基本创建和处理方法。

> 尽管一个句柄变量可以使用任意的名称，但为了使程序可读性增强，建议在命名一个保存句柄的变量时，首先以 H 开头，然后利用几个字母描述对象的类型，然后是一个下划线，最后是用于其他描述的一个字符串。例如，我们可以用 Hf_fig 表示一个图形句柄变量，用 Ha_ax1 表示一个坐标轴对象的句柄变量，用 Ht_title 表示一个文本对象的句柄变量。当用户不知道对象的类型时，可以在 H 后面使用 x 字母，例如，Hx_obj。

31.3 对象属性

所有的对象都有一组定义其特征的属性。通过设置这些属性，用户可以调整图形显示的方式。尽管有的属性名在所有的对象中都能见到，但与每个对象类型（例如，坐标轴系，线条，表面）相关的属性都是惟一的。属性可以描述对象的诸多特性，例如对象的位置、颜色、对象类型、父对象句柄、子对象句柄以及其他特性。每个不同的对象都有其自身独立的属性，改变该对象的属性不会影响其他的相同类型的对象的属性。

对象属性由属性名和相应的属性值构成。属性名由大小写混合的字符串构成，并且字符串的第一个字母大写，例如属性名'LineStyle'代表一个线对象的线型属性。需要注意的是，属性名的大小写仅仅是为了用户使用起来方便，Matlab 在识别属性名时是不区分大小写的。另外，只要能够惟一表示属性名，用户可以使用任意长度的字符串。例如，一个坐标轴对象的位置属性可以被命名为'Position'、'position'或者'pos'。

当对象被创建时，其初始化属性值就是其默认属性值。这些默认属性值可以用下述两种方法进行改变：①在创建对象时，在函数调用中包含属性的设定，即（'PropertyName', 'PropertyValue'）。②在对象创建之后，利用相应的函数改变属性的值。下面的代码使用第一种方法改变属性的值：

```
>> Hf_1 = figure('Color','yellow')
```

上面的命令创建了一个默认属性的图形对象，但将其背景颜色设置成了黄色，而不是默认的颜色。

除了图形窗口中的菜单栏和工具条之外，Matlab 还提供了函数 inspect 用于通过一个图形用户界面查看和修改对象属性。该函数用法很简单，用户只要在命令窗口中输入 inspect(H)，就可以查看和修改对象句柄为 H 的对象的属性。

31.4 get和set

get和set函数用于获得和改变句柄图形对象的属性。其中，函数get用于返回对象的一个或多个属性值，其常用调用语法为：get(handle, 'PropertyName')，例如，下面的命令返回句柄为Hf_1的图形对象的位置属性向量：

```
>> p = get(Hf_1,'Position')
```

下面的命令则返回句柄为Hl_a的对象的颜色属性：

```
>> c = get(Hl_a,'Color')
```

函数set用于改变句柄图形对象的属性值，其调用语法为：set(handle, 'PropertyName', PropertyValue)。例如，下面的代码将句柄为Hf_1的图形对象的位置设置为由向量p_vect所指定的位置：

```
>> set(Hf_1,'Position',p_vect)
```

下面的命令将句柄名为H1_a的对象的颜色设置为红色：

```
>> set(Hl_a,'Color','r')
```

函数set可以同时设置多个属性的值，如下例所示：

```
>> set(Hl_a,'Color',[1 0 0],'LineWidth',2,'LineStyle','--')
```

该命令将句柄名为H1_a的线条对象的颜色设为红色，线条宽度设为两个像素点宽，线型设置为虚线。

除了上述主要用途外，函数get和set还可以用于返回某一对象属性的反馈信息。例如，set(handle, 'PropertyName')将返回一列值，这列值包含所有handle对象的PropertyName属性可以被赋予的值。例如，下面的代码返回Hf_1图形对象的'Units'属性的6个可用的字符串值，并且注明'pixels'是默认值：

```
>> set(Hf_1,'Units')
[ inches | centimeters | normalized | points | {pixels} | characters ]
```

如果在上面的代码中，用户指定的'propertyName'属性没有固定的序列值，Matlab就会向用户发出一个相应的信息，如下例所示：

```
>> set(Hf_1,'Position')
A figure's 'Position' property does not have a fixed set of property
values.
```

除了set命令，句柄图形对象的其他生成函数也可以接受多个属性参数对。如下面的代码：

```
>> figure('Color','blue','NumberTitle','off','Name','My Figure')
```

生成了一个新的图形对象，并将其背景设为蓝色，标题设为'My Figure'，而不是默认的窗口标题'Figure 1'。

为了更好地理解上面的概念，请大家看下面的例子：

```
>> Hf_fig = figure  % create a figure
Hf_fig =
       1
>> Hl_light = light % add default light to an axes in the figure
Hl_light =
         107
>> set(Hl_light)    % find settable properties of light
       Position
       Color
       Style: [ {infinite} | local ]
       ButtonDownFcn: string -or- function handle -or- cell array
       Children
       Clipping: [ {on} | off ]
       CreateFcn: string -or- function handle -or- cell array
       DeleteFcn: string -or- function handle -or- cell array
       BusyAction: [ {queue} | cancel ]
       HandleVisibility: [ {on} | callback | off ]
       HitTest: [ {on} | off ]
       Interruptible: [ {on} | off ]
       Parent
       Selected: [ on | off ]
       SelectionHighlight: [ {on} | off ]
       Tag
       UIContextMenu
       UserData
       Visible: [ {on} | off ]

>> get(Hl_light)   % get all properties and names for light
       Position = [1 0 1]
       Color = [1 1 1]
       Style = infinite
       BeingDeleted = off
       ButtonDownFcn =
       Children = []
       Clipping = on
       CreateFcn =
       DeleteFcn =
       BusyAction = queue
       HandleVisibility = on
       HitTest = on
       Interruptible = on
       Parent = [108]
       Selected = off
       SelectionHighlight = on
       Tag =
       Type = light
       UIContextMenu = []
       UserData = []
       Visible = on
```

上例利用光照对象演示了set函数和get函数的用法，是因为在所有的Matlab对象中，光照对象的属性个数最少。在上面的例子中，首先生成了一个图形对象并且返回其句柄；然后生成了一个光照对象并返回对象的句柄。由于光照对象是坐标轴对象的子对象，因此在生成光照对象的同时会伴随生成一个坐标轴对象，并且该坐标轴对象的句柄可以从光照对象的'Parent'属性中获得。

从get和set函数的返回结果可以看出，对象的属性在显示时被分成了两组。第一组属性是只有该对象才具有的专有属性，第二组属性则是所有对象都具有的通用属性。另外，set和get函数所返回的属性列表也有些许差异：set函数只显示出了可以用set命令改变的属性，而get函数显示出了所有的属性。例如，在上边的例子中，由于'Type'属性是光照对象的一个属性但不能被set函数改变（'Type'属性是只读属性），所以在set函数显示的属性列表中没有这一属性，而get函数显示的属性列表中存在这一属性。

在Matlab的各个版本中，每个对象类型的属性个数是固定的，但不同的对象类型具有不同个数的属性。例如，上边的例子中，光照对象有3个专有属性和18个通用属性（共21个属性）；而坐标轴对象则多达103个属性。很明显，要将所有对象的每个属性都描述清楚或显示出来需要巨大的篇幅，同时也超出了本书的范畴。

下面我们通过一个简单的例子介绍一下对象句柄的用法：假设我们要用非标准颜色绘制一条正弦曲线，该曲线的颜色被设定为RGB值等于[1 0.5 0]（即一种中度的橙色），代码如下：

```
>> x = -2*pi:pi/40:2*pi;          % create data
>> y = sin(x);                    % find sine of x
>> Hls_sin = plot(x,y)            % plot sine and save lineseries handle
Hls_sin =
   59.0002
>> set(Hls_sin,'Color',[1 .5 0],'LineWidth',3)% Change color and width
```

如果我们要在同一幅图中添加一条淡蓝色（RGB值为[0.75 0.75 1]）的余弦曲线，可用下面的代码：

```
>> z = cos(x);                    % find cosine of x
>> hold on                        % keep sine curve
>> Hls_cos = plot(x,z);           % plot cosine and save lineseries handle
>> set(Hls_cos,'Color',[.75 .75 1])    % color it light blue
>> hold off
```

如果能够正确地使用句柄，上面两条曲线的绘制其实可以使用下面的3行代码实现：

```
>> Hls_line = plot(x,y,x,z);     % plot both curves and save handles
>> set(Hls_line(1),'Color',[1 .5 0],'LineWidth',3)
>> set(Hls_line(2),'Color',[.75 .75 1])
```

我们还可以给上面的图形添加一个标题，并使标题的字体比正常的字体大（字体大小为16）：

```
>> title('Handle Graphics Example')          % add a title
>> Ht_text = get(gca,'Title')                % get handle to title
>> set(Ht_text,'FontSize',16)                % customize font size
```

上面的例子展示了坐标轴对象的一个非常有趣的特点。在 Matlab 中，每个对象都有一个'Parent'属性和一个'Children'属性，这些两个属性分别包含了该对象的父对象和子对象的句柄。例如，对于一个绘制在某一坐标系中的线条对象而言，它的'Parent'属性包含了坐标轴对象的句柄，'Children'属性包含了一个空数组（即它目前还没有子对象）；而对于坐标轴对象而言，'Parent'属性包含了该坐标轴所在的图形的句柄，'Children'属性则包含了该坐标轴中线条对象的句柄。文本类型的对象与坐标轴之间的继承关系不像线条对象那样一目了然：用 text 和 gtext 命令创建的文本对象是坐标轴对象的子对象，它们的句柄也都包含在坐标轴对象的'Children'属性中；而与标题字符串和坐标轴标签相关的对象句柄却没有在'Children'属性中体现，它们位于坐标轴对象的'Title'，'XLabel'，'Ylable'和'ZLable'属性中，并且在创建坐标轴时便自动生成。虽然前面的例子中 title 函数仅仅设置了当前坐标轴中标题文本对象的'String'属性，但实际上标准的 Matlab 文本函数如 title、xlable、ylabel、zlabel 等都能接受一个或多个属性和属性值参数对，并且返回一个指向该文本对象的句柄。例如，下面的命令给当前图形添加了一个大小为 24 像素点的绿色标题，并且返回这个标题文本对象的句柄：

```
>> Ht_title = title('This is a title.','FontSize',24,'Color','green')
```

除了 set 和 get 函数之外，Matlab 还提供了其他几个函数用于处理对象和对象的属性。例如，用户可以使用 copyobj 函数将某个对象从一个父对象拷贝到另一个父对象。我们看下面的语句：

```
>> Ha_new = copyobj(Ha_ax1,Hf_fig2)
```

上面的语句将句柄名为 Ha_ax1 的坐标轴对象及其所有子对象拷贝到句柄名为 Hf_fig2 的图形中，并给拷贝的坐标轴对象分配一个新的句柄 Ha_new。只要满足本书第一节的分级图中的对象继承关系，任何对象都可以被拷贝到任何合法的父对象中。copyobj 函数既可以包含一个参数，也可以包含两个参数，但必须为句柄向量。

另外，用户也可以通过将一个对象的'Parent'属性设置为另一个合法的父对象句柄，从而将该对象从其原来的父对象转移到另一个父对象。注意，上述过程是一个搬移的过程，而不是拷贝的过程，也就是说，原来的父对象中不再包含被搬移的对象。我们看下面的例子：

```
>> figure(1)
>> set(gca,'Parent',2)
```

上述命令将当前坐标轴对象及其所有子对象从句柄为 1 的图形搬移到句柄为 2 的图形中。搬移过程不会对图形 2 中的已有对象产生影响，但这些对象可能会被新的坐标轴对象遮掩。

用户可以利用 delete(handle)函数删除某一对象及其所有子对象。另外，用户也可以使用 reset(handle)函数将句柄名为 handle 的对象的属性（除'Position'属性之外）值重置为默认值。在 set、reset、copyobj 和 delete 函数调用时，如果参数 handle 是对象句柄的一个列向量，则这些函数能够影响所有被列出的对象。

在前面的例子中，除了可以利用函数 get 和 set 直接观察对象的属性列表外，还可以将该列表返回到一个结构体变量中，请看下面的例子：

```
>> lprop = get(Hl_light)
lprop =
                 BeingDeleted: 'off'
                   BusyAction: 'queue'
                ButtonDownFcn: ''
                     Children: [0×1 double]
                     Clipping: 'on'
                        Color: [1 1 1]
                    CreateFcn: ''
                    DeleteFcn: ''
             HandleVisibility: 'on'
                      HitTest: 'on'
                Interruptible: 'on'
                       Parent: 108
                     Position: [1 0 1]
                     Selected: 'off'
           SelectionHighlight: 'on'
                        Style: 'infinite'
                          Tag: ''
                         Type: 'light'
                UIContextMenu: []
                     UserData: []
                      Visible: 'on'
>> class(lprop) % class of get(Hl_light)
ans =
struct

>> lopt = set(Hl_light)
lopt =
                   BusyAction: {2×1 cell}
                ButtonDownFcn: {}
                     Children: {}
                     Clipping: {2×1 cell}
                        Color: {}
                    CreateFcn: {}
                    DeleteFcn: {}
             HandleVisibility: {3×1 cell}
                      HitTest: {2×1 cell}
                Interruptible: {2×1 cell}
                       Parent: {}
                     Position: {}
                     Selected: {2×1 cell}
           SelectionHighlight: {2×1 cell}
                        Style: {2×1 cell}
                          Tag: {}
                UIContextMenu: {}
                     UserData: {}
                      Visible: {2×1 cell}
```

```
>> class(lopt) % class of set(Hl_light)
ans =
struct
```

上述语句中，返回的结构体的各个域名就是对象的各个属性名字符串，并按照字母顺序排列。请注意，尽管属性名是不区分大小写的，但是结构体中的域名却是区分大小写的，如下例所示：

```
>> lopt.BusyAction
ans =
    'queue'
    'cancel'
>> lopt.busyaction
??? Reference to non-existent field 'busyaction'.
```

既然属性列表被保存在了结构体中，用户就可以利用结构体语法来设置一个或多个属性的值，如下例所示：

```
>> newprop.Color = [1 0 0];
>> newprop.Position = [-10 0 10];
>> newprop.Style = 'local';
>> set(Hl_light,newprop)
```

上述语句改变了光照对象的'Color'、'Position'和'Style'属性的值，但对其他属性没有任何影响。请注意，用户不能在将属性列表返回到一个结构体后，直接用该结构体重新设置属性的值，如下例所示：

```
>> light_prop = get(Hl_light);
>> light_prop.Color = [1 0 0];      % change the light color to red
>> set(Hl_light,light_prop);        % reapply the property values
??? Error using ==> set
Attempt to modify read-only light property: 'Type'.
```

从上面的错误信息可以看出，由于'Type'是光照对象仅有的只读属性，因此，用户不能设置整个结构体包含的属性。用户可以通过将'Type'域从结构体中删除的方法来解决这个问题，如下例所示：

```
>>light_prop = rmfield(light_prop,'Type','BeingDeleted');
>> set(Hl_light,light_prop)
```

对于包含更多只读属性的对象，只有当所有的只读属性都从结构体中删除后，用户才能使用这个结构体来设置属性值。

除了结构体外，单元数组也可以用来查询属性值。为此，用户需要首先创建一个单元数组，该单元数组要包含需要查询的属性名，各属性名需要按一定顺序排列；然后将这个单元数组传递给 get 函数，get 函数返回的结果也是一个单元数组，如下例所示：

```
>> plist = {'Color','Position','Style'}
plist =
    'Color'  'Position'  'Style'
>> get(Hl_light,plist)
```

```
ans =
    [double]    [1×3 double]    'local'
>> class(ans)   % cell array in, cell array out
ans =
cell
```

关于get函数，还有一点需要用户注意：如果H是一个句柄向量，则get(H, 'PropertyName')将返回一个单元数组而不是一个向量，请看下面的例子（假定gcf为一个包含4个子图的图形窗口句柄）：

```
>> Ha = get(gcf,'Children')              % get axes handles
Ha =
    15.0002
    13.0002
    11.0002
     9.0002
>> Ha_kids = get(Ha,'Children')          % get handles of axes children
Ha_kids =
    [    16.0002]
    [4×1 double]
    [    12.0002]
    [2×1 double]
>> class(Ha_kids)
ans =
    cell
>> Hx = cat(1,Ha_kids{:})                % convert to column vector
Hx =
    16.0002
    26.0002
    24.0002
    18.0002
    22.0002
    12.0002
    14.0002
    10.0002
>> class(Hx)
ans =
    double
```

这样，Hx 就可以作为一个对象句柄向量传递给需要以句柄向量为参数的句柄图形函数。

31.5 查找对象

如前所述，句柄图形向用户提供了一个访问图形窗口中对象的方法，并使得用户可以利用get和set函数定制图形。不过，前面讲到的函数都需要用户提供进行处理的对象的句柄。当用户不知道一个对象的句柄时，可以使用Matlab提供的句柄查询函数来寻找对象的

句柄。其中 gcf 和 gca 函数就是较早的引入的两个句柄查询函数。gcf 函数用于返回当前图形窗口的句柄，例如：

```
>> Hf_fig = gcf
```

gca 函数用于返回当前图形窗口中当前坐标轴的句柄，例如：

```
>> Ha_ax = gca
```

除了上面两个函数之外，Matlab 还提供了一个函数 gco 用来获得当前对象的句柄，例如：

```
>> Hx_obj = gco
```

gco 函数也可以带有一个句柄参数，表示返回指定的图形中当前对象的句柄，例如：

```
>> Hx_obj = gco(Hf_fig)
```

将返回句柄名为 Hf_fig 的图形中当前对象的句柄。这里的“当前对象”被定义为指定图形中鼠标所点击的最后一个对象，这个对象可以是除了根对象之外的任何对象。当一个图形被刚刚创建时，不存在当前对象，因此 gco 将返回一个空数组。只有当鼠标光标落在图形区域内并点击了鼠标后，gco 函数才返回一个对象（即被点击的对象）的句柄。

获得了对象的句柄之后，用户就可以通过查询该对象的'Type'属性来获得该对象的类型，'Type'属性的属性值是一个诸如'figure'、'axes'或'text'等内容的字符串。'Type'属性对所有的对象都是通用的。例如，下面的代码返回 Hx_obj 对象的类型，返回值 x_type 是一个字符串：

```
>> x_type = get(Hx_obj,'Type')
```

有时候用户除了需要知道诸如'CurrentFigure'、'CurrentAxes'或'CurrentObject'等标识的当前句柄之外，还需要知道一个对象的子对象的句柄向量，这时可以使用 get 函数完成，如下例所示：

```
>> Hx_kids = get(gcf,'Children');
```

该语句返回当前图形的所有子对象的句柄构成的向量。

为了简化句柄查询过程，Matlab 专门提供了一个内置函数 findobj，它返回给定属性值的对象的句柄。例如，Hx=findobj(Handles, 'flat', 'PropertyName',PropertyValue)将返回 Handles 中'PropertyName'属性值为 PropertyValue 的所有对象的句柄。当然，findobj 函数也支持多个('PropertyName',PropertyValue)参数对，这时该函数返回的对象要满足每一个给定的属性和属性值条件。如果在 findobj 函数中，Handles 参数被省略，则假定该函数在根对象中查找。如果 findobj 函数没有包含('PropertyName',PropertyValue)参数对，则将返回 handles 中的所有对象的句柄。如果 findobj 函数中没有'flat'参数，则将在 handles 中的所有对象、对象的子对象（包括坐标轴标题和标签）中进行搜索。如果 findobj 函数没有找到符合条件的对象，将返回一个空矩阵。例如，下面的代码查找所有颜色为绿色的线条对象的句柄：

```
>> Hl_green = findobj(0,'Type','line','Color',[0 1 0]);
```

用户可以使用'HandleVisibility'属性（该属性是所有对象的一个通用属性）来隐藏指定对象的句柄。将对象的句柄隐藏起来，可以避免用户对对象属性的不经意删除或修改。因为当一个对象的'HandleVisibility'属性被设置为'off'或'callback'时，在 Command 窗口利用 findobj 函数查询时就不会返回该对象的句柄，另外，被隐藏的对象句柄不会出现在子对象列表中，也不会作为 gcf、gca 或者 gco 的输出。不过，当'HandleVisibility'属性被设置为'callback'时，该对象可以被回调函数发现（关于回调函数的信息请参见 31.16 节）。

31.6 用鼠标选择对象

前面讲到，gco 函数返回当前对象（即鼠标最后点击的对象）的句柄。当鼠标点击在多个对象的交叉点附近时，Matlab 需要使用一些规则来确定哪个对象是当前对象（即是鼠标点击的对象）。我们知道，每个对象都有自己的有效区域，当鼠标点击在这个有效区域内，就表明选中了这个有效区域所对应的对象。Matlab 对各个对象的有效区域的定义如下：①对于线条对象，其有效区域包括线条本身，以及距离这个线条 5 个像素点范围内的区域（如果这个区域存在的话）。②对于表面、碎片或文本对象，其有效区域是能够包含这个对象的最小的矩形区域。③对于坐标轴对象，其有效区域是坐标轴框加上标签和标题所确定的区域。在鼠标进行选择时，各个对象被选中的优先级（叠放次序）不同：如果线条和表面位于一个坐标轴内部，则这些线条和表面（包括碎片、文本）将具有比坐标轴高的被选优先级，也就是说，当鼠标点中线条或表面时，选中的是相应的线条或表面对象而不是坐标轴。最后，点中坐标轴有效区域以外的区域，就可以选中整个图形。

如果线条对象、表面对象之间具有相同的有效区域，并且当鼠标点击在这个共同有效区域内时，则需要使用叠放次序决定哪个对象为当前对象。叠放次序决定了哪个对象重叠在其他对象的上面。在对象刚创建时，叠放次序由对象的生成顺序决定，即最后生成的对象位于叠放次序的最上层。例如，如果用户输入两个 figure 命令，则第二个 figure 命令生成的图形将被显示在第一个图形的上面，其叠放次序是图形 2 在图形 1 上面，这样，gcf 函数所返回的图形句柄就是 2。如果用户在命令窗口中输入了 figure(1)命令，或者用鼠标点击了图形 1，那么就人为地将叠放次序改变，图形 1 就移到了叠放次序的最上层，成为了当前图形。

对于 figure 窗口而言，叠放次序很容易理解。但当应用到线条对象（包括表面对象）时，情况就稍有不同。比如我们在绘制两条线条时，所绘制的第二条线条和第一条线条在相交处可以看出是第二条线条位于第一条上面。但如果用鼠标点击了第一条线条，第一条线条虽然变成了当前对象，但叠放次序并不会改变（即第一条线条仍位于第二条线条的下面）。因此，除非用于显式地改变线条的叠放次序，否则用鼠标点击两线的交点将始终选择第二条线条。

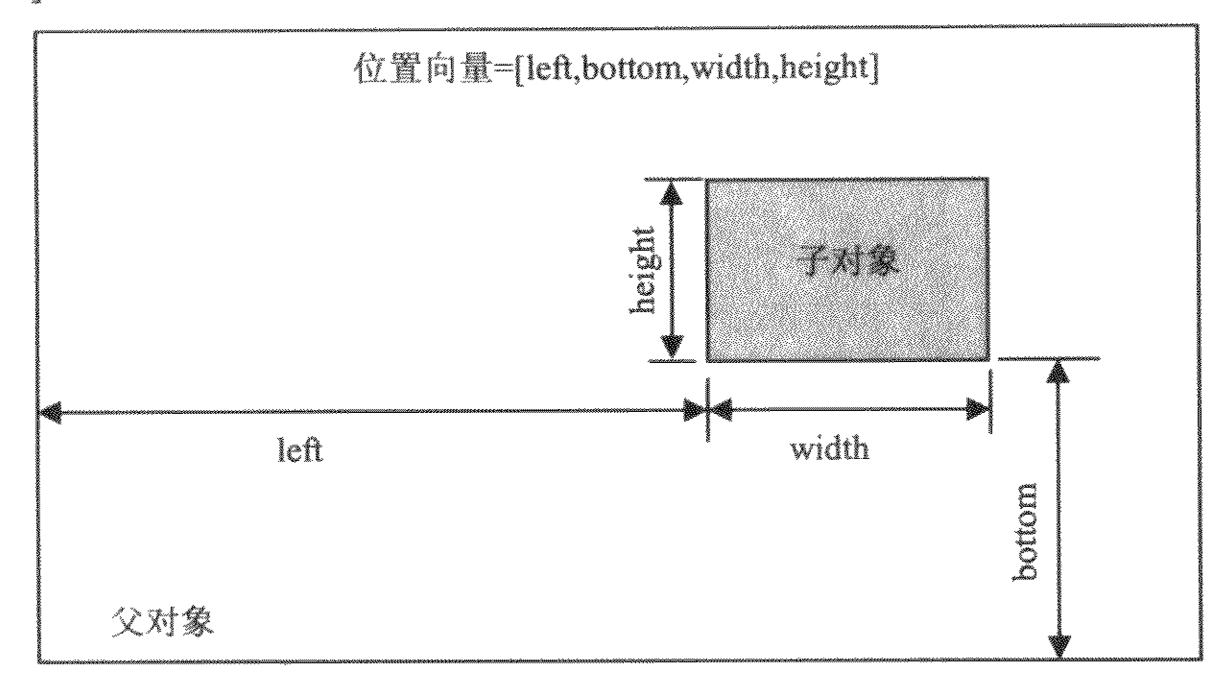
在句柄图形中，叠放次序是由一个给定对象的'Children'句柄中各子对象句柄的出现顺序决定的。也就是说，Hx_kids=get(handle,'Children')将按照叠放次序返回各个子对象的句柄。向量 Hx_kids 的第一个元素位于叠放次序的最上层，而最后一个元素位于叠放次序的最底层。用户可以通过改变一个对象的'Children'属性值的元素顺序来改变其中的子对象的叠放次序，例如：

```
>> Hf = get(0,'Children');
>> if length(Hf)>1
    set(0,'Children',Hf([end 1:end-1]);
end
```

上述命令将最底层的图形移到叠放次序的最上层，并使之成为当前图形。

31.7 位置和单位属性

不论图形（figure）对象，还是其他的句柄图形对象（如线条、表面等），其'Position'属性都是一个四元素的行向量，称为位置向量。如下图所示，位置向量中的值为[left,bottom,width,height]，其中[left,bottom]表示该对象的左下角相对于其父对象的位置，而[width,height]是该对象的宽和高。



位置向量中各值的单位是由子对象的'Unit'属性决定的，例如：

```
>> get(gcf,'Position')
ans =
    920      620      672      504
>> get(gcf,'units')
ans =
pixels
```

上述语句表明：当前图形对象的左下角位置位于距屏幕左端 920 个像素点和底端 620 个像素点处，当前图形对象的宽度为 672 像素，高度为 504 像素。注意：图形的'Position'向量表示的区域是图形内部可以绘图的区域，不包括图形窗口的窗口边框、滚动条、菜单栏或者标题栏。

Matlab 还提供了'OuterPosition'属性，用于返回包含窗口边框、滚动条、菜单栏或者标题栏在内的图形窗口的位置（称为外层位置）向量，如下例所示：

```
>> get(gcf,'Position')          % drawable position
ans =
    920      620      672      504
>> get(gcf,'OuterPosition')     % outside position
```

```
ans =
    916     616     680     531
```

第二条语句返回了图形窗口的外层边框的左侧位置、底部位置、宽度和高度。在默认情况下，无论是可绘位置向量还是外层位置向量，当窗口中的菜单栏、工具栏等增加或减少时，都会发生相应的变化。要使这些位置向量保持不变，用户可以通过设置图形对象的'ActivePositionProperty'属性来完成。当该属性设置为'Position'时，就意味着在工具栏被显示或隐藏时可绘区域的位置向量（'Positon'值）保持不变；当该属性设置为'OuterPosition'时，则意味着在工具栏显示或隐藏时外层区域的位置向量（'OuterPosition'值）保持不变。

图形的单位（'Unit'）属性的默认设置为像素，但也可以是英寸、厘米、点、字符或归一化坐标值。像素指屏幕像素，是可以在计算机屏幕上显示的最小的矩形单位。例如，如果一台计算机的显示分辨率被设置为800×600，就意味着该计算机的屏幕显示是800像素宽，600像素高。点通常用于排版标准中，1点等于1/72英寸。1字符单位指默认系统字体中一个字符的宽度。例如，1字符单位即等于默认系统字体中字母x（也可以是其他字母）的宽度。归一化坐标指介于0和1之间的坐标值。在归一化坐标系中，父对象的左下角坐标为[0,0]，右上角坐标为[1,1]。另外，英寸和厘米这两种单位大家都很熟悉，这里就不再解释了。

我们仍用前面的位置向量来演示'Units'属性，请看下面的代码：

```
>> set(gcf,'units','inches')    % INCHES
>> get(gcf,'position')
ans =
          7.9224       5.3362       5.7931       4.3448
>> set(gcf,'units','cent')       % CENTIMETERS
>> get(gcf,'position')
ans=
          20.108       13.544       14.703       11.027
>> set(gcf,'units','normalized')% NORMALIZED
>> get(gcf,'position')
ans=
        0.57438      0.51583         0.42          0.42
>> set(gcf,'units','points')    % POINTS
>> get(gcf,'position')
ans=
          570.41       384.21        417.1        312.83
>> set(gcf,'units','char')       % CHARACTERS
>> get(gcf,'position')
ans=
          153.17       38.688         112           31.5
```

当用户的显示器和显示器分辨率没有发生变化时，上述5个位置向量均表示相同的图形位置，只不过使用的单位不同而已。

坐标轴的位置向量也是一个四元素向量，其形式和图形的位置向量相同，也是[left,bottom,width,height]，但这些值是相对于坐标轴的父图形对象的左下角位置。通常，一个子对象的'Position'属性都是相对于其父对象的位置而言的。

对于计算机屏幕（根对象），其位置属性不叫'Position'，而是'ScreenSize'（因为这样读者更容易理解）。计算机屏幕的[left,bottom]总为[0,0]，[width,height]就是整个计算机屏幕的大小，它们的单位由根对象的'Units'属性值决定。

31.8 默认属性

Matlab 会为每个新创建的对象指定默认的属性值。Matlab 为每个对象内置的默认属性均是出厂时的默认属性。为了改变这些默认属性，用户必须使用 set 和 get 函数来设置和获取相应的属性值。如果用户不想用内置的默认属性创建对象，Matlab 也允许用户设置自己的默认属性。用户既可以改变单个对象的默认属性，也可以改变对象分级图中某类对象的默认属性。在创建一个对象时，Matlab 首先在父对象这一层寻找默认属性值，如果没有找到，就沿着对象层次结构向上查找，直到找到一个默认属性值或者找到内嵌的出厂默认值。

用户可以通过使用一种特殊的属性名字符串，来设置任何一个层次的对象的默认属性值。该字符串由'Default'开始，后边紧跟对象的类型名和属性名。用户在 set 命令中所使用的句柄决定了设置的默认值应用的对象范围。例如，下面的语句将所有新创建的图形对象的默认背景颜色设置为中灰色：

```
>> set(0,'DefaultFigureColor',[.5 .5 .5])
```

由于上述语句为 set 函数传递了一个 0 句柄，因此设置的默认属性适用于根对象和其所有的子对象，即所有新创建的图形都具有一个灰色的背景。

下面给出了设置默认属性值的其他一些例子：

```
>> set(0,'DefaultAxesFontSize',14)         % larger axes fonts - all figures
>> set(gcf,'DefaultAxesLineWidth',2)       % thick axis lines - this figure only
>> set(gcf,'DefaultAxesXColor','y')        % yellow X axis lines and labels
>> set(gcf,'DefaultAxesYGrid','on')        % Y axis grid lines - this figure
>> set(0,'DefaultAxesBox','on')            % enclose axes - all figures
>> set(gca,'DefaultLineLineStyle',':')     % dotted linestyle - these axes only
```

当用户改变默认属性时，只有那些在设置语句之后生成的对象才会使用新的默认属性，而已经存在的对象仍然保持原来的默认属性不变。

当用户需要对一个已经存在的对象进行操作处理时，最好在处理完后将这些对象恢复到原来的状态。例如，如果用户在一个 M 文件中改变了已存在对象的默认属性，那么最好先将该对象原来的默认属性保存起来，在处理完毕要退出程序时，再将这些对象的默认属性恢复到最初的设置，如下面的代码所示：

```
oldunits = get(0,'DefaultFigureUnits');
set(0,'DefaultFigureUnits','normalized');
    <MATLAB statements>
set(0,'DefaultFigureUnits',oldunits);
```

要想使 Matlab 在任何时候都使用用户定义的默认值，用户只需要在 startup.m 文件中包含相应的 set 命令或命令组即可，例如：

```
set(0,'DefaultAxesXGrid','on')
set(0,'DefaultAxesYGrid','on')
set(0,'DefaultAxesZGrid','on')
set(0,'DefaultAxesBox','on')
set(0,'DefaultFigurePaperType','A4')
```

这些命令表明无论何时创建图形对象，都显示坐标栅格线，并且显示封闭的坐标轴边框，另外将默认的纸张大小设置为 A4 纸。由于这些设置都是在根一级对象进行的，因此这些设置值将会影响到图形窗口中的每一个对象。

Matlab 提供了 3 个特殊的属性值字符串用于取消、覆盖或查询用户自定义的默认属性，它们是'remove'、'factory'和'default'。例如，如果用户改变了一个对象的默认属性，可以使用特殊属性值'remove'来取消这次改动，从而将该对象的属性重新设置为它原来的默认值，例如：

```
>> set(0,'DefaultFigureColor',[.5 .5 .5])  % set a new default
>> set(0,'DefaultFigureColor','remove')    % return to MATLAB defaults
```

为了临时覆盖用户设置的默认属性，或临时在某一特定对象上使用 Matlab 出厂默认属性值，用户可以使用特殊属性值'factory'，例如：

```
>> set(0,'DefaultFigureColor',[.5 .5 .5])% set a new user default
>> figure('Color','factory')          % figure using default color
```

第三个特殊属性值字符串'default'强迫 Matlab 沿着对象层次结构向上搜索，直到找到所需要的属性默认值。如果 Matlab 能够找到这个默认值，就使用这个默认值；如果已经到达了根对象还没有找到所需的默认值，就使用 Matlab 出厂默认值。该特殊属性值在用户已经使用非默认属性值生成了对象，但希望将属性值重新设为默认属性时非常有效，例如：

```
>> set(0,'DefaultLineColor','r')        % set default at the root level
>> set(gcf,'DefaultLineColor','g')      % current figure level default
>> Hl_rand = plot(rand(1,10));      % plot a line using 'ColorOrder' color
>> set(Hl_rand,'Color','default')       % the line becomes green
>> close(gcf)                           % close the window
>> Hl_rand = plot(rand(1,10));% plot a line using 'ColorOrder' color again
>> set(Hl_rand,'Color','default')       % the line becomes red
```

注意：plot 函数在正常情况下，不会使用线条对象的默认值作为其绘制的线条颜色。如果用户没有在 plot 函数的参数中指定颜色，那么 plot 命令就用坐标轴的'ColorOrder'属性值来设置它绘制的每一条线条的颜色。

要想获得所有的出厂默认属性值列表，可以使用下面的命令：

```
>> get(0,'factory')
```

要想获得对象层次结构中任何一层所设置的默认属性，可以使用下面的命令：

```
>> get(handle,'default')
```

根对象包含了大量颜色属性的默认值以及图形初始创建位置属性的默认值，如下所示：

```
>> get(0,'default')
ans =
```

```
            defaultTextColor : [0 0 0]
           defaultAxesXColor : [0 0 0]
           defaultAxesYColor : [0 0 0]
           defaultAxesZColor : [0 0 0]
       defaultPatchFaceColor : [0 0 0]
       defaultPatchEdgeColor : [0 0 0]
            defaultLineColor : [0 0 0]
defaultFigureInvertHardcopy : 'on'
          defaultFigureColor : [0.8 0.8 0.8]
            defaultAxesColor : [1 1 1]
       defaultAxesColorOrder : [7×3 double]
       defaultFigureColormap : [64×3 double]
     defaultSurfaceEdgeColor : [0 0 0]
       defaultFigurePosition : [920 620 672 504]
```

其他默认属性只有在被用户设置之后才会显示在列表中，如下所示：

```
>> get(gcf,'default')
ans =
0x0 struct array with fields:
>> set(gcf,'DefaultLineMarkerSize',10)
>> get(gcf,'default')
ans =
    defaultLineMarkerSize:10
```

31.9 通用属性

通用属性是所有的句柄图形对象都具有的一组属性。下表给出了所有的通用属性的属性名及其描述：

属性	描述
BeingDeleted	该属性标示对象是否能被删除。只有该属性设置为'on'时，用户才可以删除对象
BusyAction	该属性用于控制 Matlab 句柄如何回调中断
ButtonDownFcn	该属性指定了当鼠标在一个对象上按下时，需要执行的回调代码
Children	该属性返回所有可见的子对象句柄
Clipping	该属性用于激活或者禁用对坐标轴子对象的范围限制（即是否能够超出坐标轴范围）
CreateFcn	该属性指定了在一个对象被创建之后需要立即执行的回调代码
DeleteFcn	该属性指定了在一个对象被删除之前需要执行的回调代码
BusyAction	决定该对象的回调过程如何被其他回调过程中断
HandleVisibility	决定该对象的句柄是否在命令窗口中或执行回调时可见
HitTest	决定该对象是否能够用鼠标选定并成为当前对象
Interruptible	决定该对象的回调是否可以被中断
Parent	该属性返回所有可见的父对象句柄
Selected	确定该对象是否已经被选为当前对象

（续表）

属性	描述
SelectionHighlight	确定该对象在选定时是否显示可见的选择句柄
Tag	该属性是一个用户自定义的字符串，用来标识对象或给对象添加一个标签。该属性通常用于 findobj 函数，例如，findobj(0, 'tag', 'mytagstring')
Type	该属性是一个标识对象类型的字符串
UIContextMenu	返回与该对象有关的上下文菜单的句柄
UserData	储存与该对象有关的所有用户自定义的变量
Visible	该属性标示对象是否可见

在上表的属性中，有 3 个属性包含了回调过程：'ButtonDownFcn'、'CreateFcn'和'DeleteFcn'。回调就是当对象属性所描述的动作发生时需要执行的 Matlab 代码。在大多数情况下，这些代码都是以函数的形式出现的。'Parent'和'Children'属性均包含了对象层次结构中其他对象的句柄。'Clipping'属性用来设置在坐标轴子对象显示时，是否进行范围限制。除文本对象之外，当'Clipping'为'on'（'on'为'Clipping'的默认值）时，坐标轴的其他子对象在显示时都可能会被削减，以便使它们显示在有效的坐标轴范围之内。'Interruptible'和'BusyAction'属性用于控制当前一个回调正在执行时，如何执行下一个回调。'Type'是一个标识对象类型的字符串。当一个对象是图形的'CurrentObject'时，则该对象的'Selected'属性值就为'on'，'SelectionHightlight'属性决定了在该对象被选中时是否改变外观。'HandleVisibility'属性用来指定对象句柄是可见的、不可见的或者只在回调时可见。如果有必要，根对象的'ShowHiddenHandles'属性将覆盖所有子对象的'HandleVisibility'属性。如果对象的'Visible'属性被设置为'off'，那么该对象将不会被显示出来（但该对象仍存在于原来的位置，并且它的对象句柄仍旧有效），并且它不会被重新绘制。将'Visible'设置为'on'将会使该对象在屏幕上显示出来。'Tag'和'UserData'属性是为用户保留的，'Tag'属性通常用来给一个对象加上标示性的标签，例如：

```
>> set(gca,'Tag','My Axes')
```

上面的语句给图形中的当前坐标轴添加一个标签'My Axes'。注意：这个字符串并不显示在坐标轴或者当前图形中，但用户可以通过查询'Tag'属性来标识这个对象。例如，在有多个坐标轴时，用户可以通过输入如下的语句找出'My Axes'坐标轴对象的句柄：

```
>> Ha_myaxes = findobj(0,'Tag','My Axes');
```

'UserData'属性可以包含任何用户想要放置的变量。字符串、数字、结构体、甚至多维单元数组都可以保存在对象的'UserData'属性中。Matlab 没有提供函数来改变或预设该属性中所包含的值，这些值只能靠用户设定和改变。

前面用 get 和 set 函数所列出的各个对象的属性都是存档属性。也有一些 Matlab 的开发人员使用的无存档或隐藏属性。这些属性其中一部分可以进行修改，而另一部分则是只读的。无存档属性虽然不能在 get 或 set 的显示列表中显示，但这些属性的确是存在的，并且可以进行修改。用户可以使用根对象的'HideUndocumented'属性（该属性本身也是一个非正式属性）控制 get 函数是返回所有的属性还是只返回正式属性，例如：

```
>> set(0,'HideUndocumented','off')
```

显然，上述语句使得非正式属性在 Matlab 中是可见的。

由于无存档属性是有意不存档的，因此用户在使用它们的时候务必要小心。无存档属性有时不如存档属性那样具有很强的鲁棒性，并且经常发生变化。随着 Matlab 版本的不断升级，无存档属性在以后也可能会继续存在、消失、发生功能改变、甚至变成存档属性。

31.10 绘制（PLOT）对象

前文已经讲过，Matlab 7 新增了绘制（PLOT）对象和组（GROUP）对象。其中，绘制对象是与 26、27 章介绍的高级图形函数相关联的对象。在 Matlab 7 中，绘制对象提供了一种将由高级图形函数创建的各个内核对象组合在一起的方法，这样的话，这些内核对象的总体属性就可以很容易进行表示和修改。也就是说，绘制对象具有单独的内核对象不具有的一些附加属性，这些附加属性通常用于指定由高级绘图函数创建的图形的属性。下面我们通过一个条形图分析绘制对象的用法，代码如下：

```
>> hbs_mm7 = bar(randn(1,6));
>> title('Figure 31.1 Random Bar Graph')
>> get(hbs_mm7,'type')
ans =
hggroup
>> hx_hbs = get(hbs_mm7,'children');
>> get(hx_hbs,'type')
ans =
patch
```

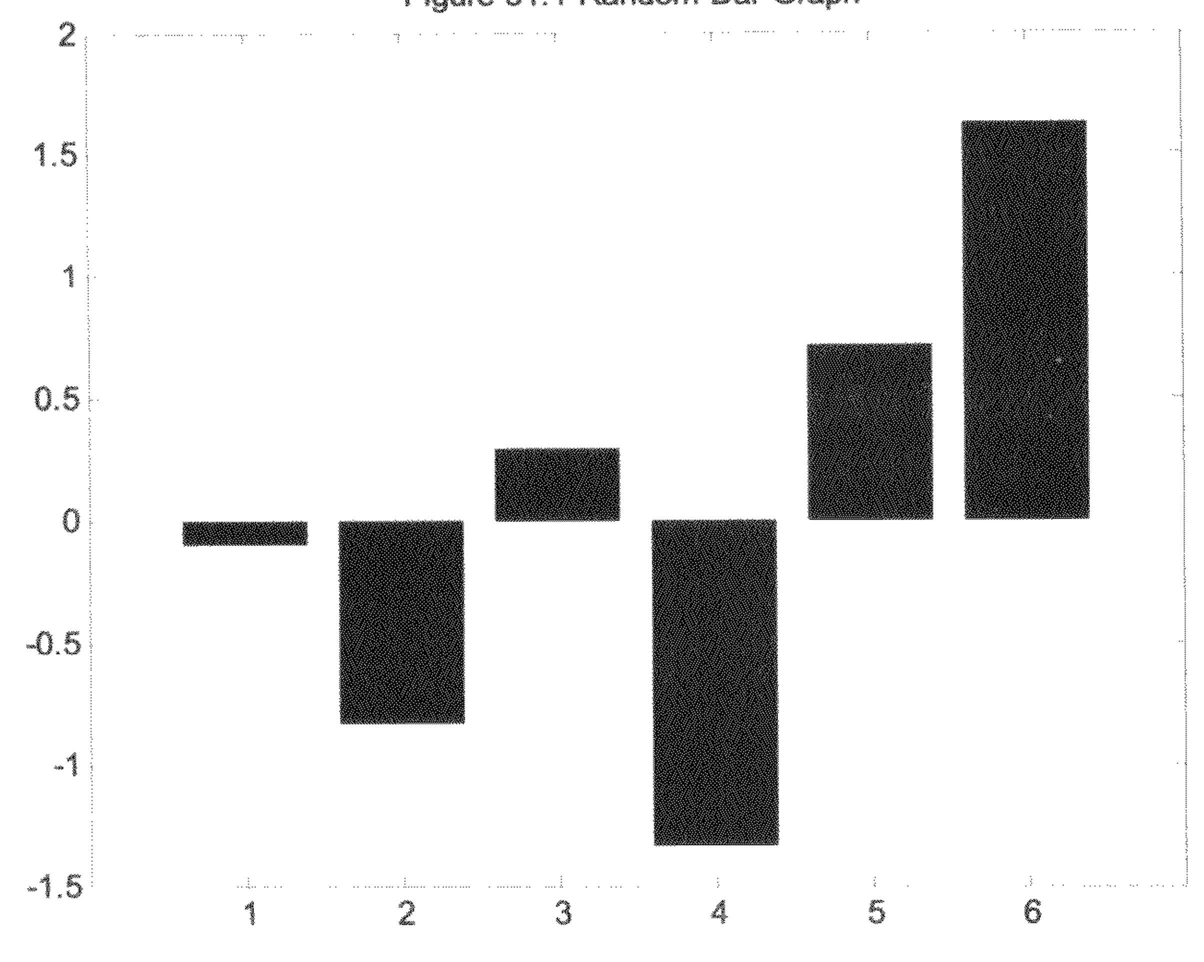


图 31.1　随机条形图

图 31.1 中的条形图是一个由复合条形序列构成的绘制对象，该对象的句柄类型被记为 hggroup。其中的每一个单独的条形碎片对象都是绘制对象的子对象。绘制对象并不包含碎片对象的所有属性，而只包含了直接表示 bar 函数所创建的图表的一些属性，例如'BaseLine'、'BarLayout'和'BarWidth'。不过，改变这些属性也将会改变相应的碎片对象的属性。将绘制对象及其属性置于碎片对象之上，将会在很大程度上方便用户处理条形图表，因为用户不需要考虑如何处理碎片属性就可以改变图表的外观。

下表给出了 Matlab 提供的绘制对象的列表，其中第二列为当用户调用 get(H, 'type')时，返回的对象类型属性值，其中 H 为相应的绘制对象的句柄。

绘制对象	get(H,'type')	能够创建该对象的高级图形函数
areasseries（区域系列对象）	hggroup	area
barseries（条形系列对象）	hggroup	bar、bar3、bar3h
contourgroup（等高线系列对象）	hggroup	contour、contour3、contourf
errorbarseries（误差条形系列对象）	hggroup	errorbar
lineseries（线条系列对象）	line	plot、plot3、semilogx、semilogy、loglog
quivergroup（箭头图组对象）	hggroup	quiver、quiver3
scattergroup（散布点系列对象）	hggroup	scatter、scatter3
stairseries（梯形图系列对象）	hggroup	stairs
stemseries（棒棒图系列对象）	hggroup	stem、stem3
surfaceplot（表面绘制对象）	surface	surf、mesh

在 Matlab 7 没有出现以前，上表中的各个高级图形函数实际上返回的是内核对象的句柄，因此，为了与老版本的 Matlab 兼容，Matlab 7 为这些高级图形函数提供了一个特殊的参数'v6'。当用户在高级绘图函数中将'v6'用做第一个输入参数时，例如，H1=plot('v6',…)，将返回各个线条对象（这是个内核对象）的句柄向量，而不是线条系列对象（这是个绘制对象）的句柄。

至于内核对象，由于存在太多的有关绘制方面的属性，这里就不再一一介绍，有兴趣的读者可以参考相应的 Matlab 帮助文档。

31.11 组（GROUP）对象

前面介绍的绘制对象从本质上讲是组对象的一个应用实例。高级图形函数所返回的每一组（或系列）绘制对象从对象层次结构上讲都是介于用户的图形对象和底层内核对象之间的一个对象层。也就是说，每个绘制组都是内核函数的父对象，也都是用户图形对象的子对象。为了增强绘制对象的功能，Matlab 7 提供了函数 hggroup，用来实现绘制对象的分组处理能力。比如，用户可以将任意数量的坐标轴对象的子对象组织到一个组中，这些子对象可以是任何的内核对象、绘制对象或其他利用 hggroup 函数封装的组对象。因此，组对象的父对象是一个坐标轴对象或者是另一个组对象。

通过将多个句柄图形对象分组，并为其指定一个单独的组句柄，那么一组对象的可视

性和可选性就可以进行统一设置。也就是说，如果设置了组对象的'visible'属性，就意味着同时设置组中所有对象的'visible'属性。另外，用鼠标单击组对象中任何一个成员对象，就可以选中整个组对象。尽管Matlab创建的绘制对象具有与其底层内核对象不同的附加属性，但组对象并没有该性质，新创建的组对象不具有与成员对象不同的特殊属性。组对象具有31.9 节给出的所有通用属性，另外，还具有'EraseMode'属性用于控制组中所有内核对象的'EraseMode'属性。

在 Matlab 7 中，创建组对象的函数是 hggroup。下面的代码给出了创建组对象的一个例子：

```
Ha = newplot;                          % create new axes object

Hl = line(1:6,[rand(1,6);1:6]);        % create 2 core line objects
Hp = patch([3 5 4],[2 2 5],'w');       % create a core patch object

Hg_mm7 = hggroup;       % create a new group object and return handle

set(Hl,'Parent',Hg_mm7)                % place lines in group
set(Hp,'Parent',Hg_mm7)                % place patch in group

get(Hg_mm7)                            % look at group properties
    EraseMode = normal
    HitTestArea = off

    BeingDeleted = off
    ButtonDownFcn =
    Children = [(3 by 1)double array]
    Clipping = on
    CreateFcn =
    DeleteFcn =
    BusyAction = queue
    HandleVisibility = on
    HitTest = on
    Interruptible = on
    Parent = [151.017]
    Selected = off
    SelectionHighlight = on
    Tag =
    Type = hggroup
    UIContextMenu = []
    UserData = []
    Visible = on
```

在上面的返回结果中，组对象 hggroup 的子对象为两个线条对象和一个碎片对象。线条对象和碎片对象的属性被组对象的属性所隐藏（因为设置组对象的属性就会同时设置所有成员对象的相应属性）。另外，组对象的'Type'属性为 hggroup。

很多时候，用户需要（左右）旋转、（前后）翻转或缩放整个组对象，但组对象的属性列表中并没有提供这些功能，不过，用户可以使用 hgtransform 函数来完成。与 hggroup 函数类似，hgtransform 函数也创建一个组对象，并能将所有的坐标轴对象及其子对象变为自己的子对象，其中包括内核对象、绘制对象和其他一些可以用 hgtransform 函数分组的对象。

利用 hgtransform 函数创建的组对象除了具有 31.9 节的所有通用属性外，还具有一个'Matrix'属性，用于指定应用到组中所有成员对象上的图形变换矩阵。变换矩阵的内容将根据标准图形转换规则创建。使用 makehgtform 函数可以使变换矩阵的创建过程变得很方便。

31.12 注释坐标轴

注释坐标轴是一种句柄可视化属性为 off 的内核坐标轴对象。该对象包含了通过使用窗口中提供的一系列菜单、工具条、面板、浏览器和编辑器，向图形中添加的所有注释内容。该坐标轴对象的'units'属性被设置为'normalized'，位置向量为[0 0 1 1]，这样对象的范围将会覆盖整个图形区域。直线、箭头线、矩阵、文本框等既可以使用图形窗口中的交互式工具创建，也可以使用 annotation 函数创建。由于添加到图形中的注释存在于一个独立的坐标轴对象中，因此，它们不会随着图形中底层图形的添加或修改而改变。因此，建议读者在绘制完底层图形之后再创建注释坐标轴。

31.13 链接对象

有时候，用户想把若干个对象的相同属性链接起来，这样，改变任何一个对象的属性，就可以改变其他对象的相同属性。Matlab 提供了函数 linkprop 和 linkaxes 函数支持链接操作。其中，linkprop 支持提供了一般对象属性的链接，linkaxes 则是一个使用 linkprop 函数专门链接两个或多个对象的坐标轴范围属性的 M 文件函数。

下面的代码给出了 linkaxes 函数的一个具体使用方法：

```
>> H_a1 = subplot(2,1,1);      % create axes object
>> plot(rand(1,30))            % plot some random data
>> H_a2 = subplot(2,1,2);      % create a second axes
>> plot(randn(1,30))           % plot random data on second axes

>> linkaxes([H_a1 H_a2],'xy')  % link x and y axis limits
```

上述代码将两个坐标轴的坐标范围属性进行链接。如果用户在图形窗口中进行全屏或缩放操作，在一个坐标轴中使用的全屏或缩放操作也会作用在另一个坐标轴中。另外，在一个坐标轴中使用 axis 命令，将使该命令对所有的链接对象都有效。

前面的例子也可以使用 linkprop 函数完成，代码如下：

```
>> close                    % close current figure to start over
>> H_a1 = subplot(2,1,1);   % create axes object
>> plot(rand(1,30))         % plot some random data
>> H_a2 = subplot(2,1,2);   % create a second axes
>> plot(randn(1,30))         % plot random data on second axes

>> H_link = linkprop([H_a1 H_a2],{'Xlim','Ylim'})
H_link =
        graphics.linkprop

>> class(H_link)
ans =
```

```
graphics.linkprop

>> methods(H_link)
Methods for class graphics.linkprop:
addprop     addtarget      linkprop        removeprop      removetarget
```

上述代码中，linkprop 函数的返回值是名为 graphics.linkprop 的一个 linkprop 对象的句柄。该对象其实是一个使用 Matlab 的面向对象编程函数（见本书第 33 章）创建的面向对象的类。代码中还显示了面向对象类中的 5 种处理 linkprop 对象的方法函数。当用户使用 linkprop 函数将多个对象的属性链接起来后，必须保证链接返回值不被删除、破坏和覆盖。例如，如果上例中的 H_link 被用户从工作区中删除、破坏或覆盖，则创建的链接将会被解除。要修改一个 linkprop 对象，需要使用上例中显示的 5 个处理函数中的一个，并且需要为该函数传递相应的对象句柄和一个或多个属性名，如下面的代码将 z 轴的坐标范围添加到 H_link 链接对象中：

```
>> addprop(H_link,'Zlim')
```

有关链接对象的其他内容请读者参考相应的 Matlab 帮助文档。

31.14 新的图形

当用户使用 line 或 text 等底层命令创建一个新的图形对象时，该对象在默认条件下将出现在当前图形的当前坐标轴上。但是，高级图形函数，如 mesh 和 plot，在创建一个图形之前会将当前的坐标轴清除，并且将大多数坐标轴属性重新设置成它们的默认值。不过，如果用户不希望高级图形函数改变当前坐标轴的属性，可以使用 hold 命令。每一个图形和坐标轴都包含一个属性'NextPlot'，用于控制 Matlab 如何使用已有的图形和坐标轴。该属性可以被 hold、newplot、reset、clf 和 cla 等命令更改。'NextPlot'属性有 3 个可能的属性值，我们可以通过 set 函数显示如下：

```
>> set(gcf,'NextPlot')
    [ {add} | replace | replacechildren ]
>> set(gca,'NextPlot')
     [ add | {replace} | replacechildren ]
```

由上述结果可知，图形的默认'NextPlot'属性为'add'；坐标轴的默认'NextPlot'属性为'replace'。当'NextPlot'被设置为'add'时，表明不需要清除或重新设置当前的图形或坐标轴就可以添加新的图形；当'NextPlot'被设置为'replace'时，表明在添加新对象并绘制图形之前，图形或坐标轴会将所有的子对象清除，并将除'Position'和'Units'之外的所有属性重新设置为默认值。该默认设置将清除并重新设置当前坐标轴，然后重新使用当前图形。

'NextPlot'的第三个可选属性值为'replacechildren'。该设置将所有的子对象清除，但不改变当前图形或坐标轴的属性。当用户在程序中使用 hold 命令时将会影响'NextPlot'属性的设置：hold on 命令将图形和坐标轴的'NextPlot'属性设置为'add'；而 hold off 命令将坐标轴的'NextPlot'属性设置为'replace'。

newplot 函数将根据'NextPlot'属性的设置生成一个新的坐标轴。该函数主要是用来生成

能够包含诸如 line、patch 等内核对象的坐标轴，而不会包含诸如 plot、surf 等高级图形函数创建的图形对象。函数 newplot 的代码有点类似于下面的代码段。

```
function Ha = newplot
Hf = gcf;                              % get current figure or create one
next = lower(get(Hf,'NextPlot'));
switch next
    case 'replacechildren',clf;        % delete figure children
    case 'replace',clf('reset');       % delete children and reset properties
end
Ha = gca;                              % get current axes or create one
next = lower(get(Ha,'NextPlot'));
switch next
    case 'replacechildren',cla;        % delete axes children
    case 'replace',cla('reset');       % delete children and reset properties
end
```

31.15 绘图速度

在很多应用场合，用户希望显示图形的一个或多个属性每一步改变的具体情况；或希望利用鼠标动态修改一个图形图像。这时，用户都希望屏幕绘图的速度尽量最快，并且不会出现图像抖动现象。鉴于此，Matlab 的图形、坐标轴、线条、碎片、矩形和表面对象都提供了一些属性用于控制绘图速度和图像抖动。

Matlab 图形对象有 3 个影响绘图速度的属性：'Renderer'、'DoubleBuffer'和'BackingStore'。其中，'Renderer'指的使用什么底层算法在屏幕上创建或绘制图形图像。该属性有 3 个可选值：'painters'、'zbuffer'和'openGL'。在通常情况下，Matlab 会根据要绘制的图形对象的复杂度和所使用的计算机性能自动决定使用哪一种底层绘图算法，并且在大多数情况下，都会为每个图形对象选择最优的绘图算法。不过，在特定情况下，用户也可以自己指定希望使用的绘图算法。

'BackingStore'属性用于控制是否为图形窗口数据创建副本。当该属性被设置为'on'时，Matlab 会为图形窗口数据创建副本，因此，当隐藏或部分隐藏的图形窗口被选中而成为当前窗口时，这些窗口能够使用副本数据立即重绘。当该属性被设置为'off'时，Matlab 不会为图形窗口数据生成后台副本，这样可以增加绘图速度，但是，在一个隐藏或部分隐藏的图形窗口被选中而成为当前窗口时，计算机需要对图形数据进行重新计算。

图形的'DoubleBuffer'属性用于控制是否将图形窗口数据存储在后台缓冲区中。该属性通常设置为'off'，表明不将图形窗口数据存储到后台缓冲区，而是实时绘制。当用户只能用 Painter's 算法绘制图形时，就必须将该属性设置为'off'。将'DoubleBuffer'属性设置为'on'就是告诉 Matlab 将所有的图形对象首先在一个后台缓冲区中进行绘制，只有当缓冲区被完全更新时，才将图形数据一起绘制在屏幕上。通过将图形绘制在一个后台缓冲区中而不直接在屏幕上显示，就可以将绘图过程与用户隔离（即用户无法看到实时的绘图过程）。实际上，后台缓冲区绘制只对简单的不产生抖动的线条类图形有效（比如 plot 函数生成的图形），并且，该图形还必须将其'EraseMode'属性设置为默认的'normal'，而对于包含了表面或碎片的

图形，后台缓冲区绘制往往无法取得良好的效果。

当用户使用 Painter's 算法绘制坐标轴时，可以利用坐标轴的'DrawMode'属性控制绘制诸如线条这样的对象所需的幕后工作量。在默认条件下，'DrawMode'属性被设置为'normal'，表示以正常速度绘制。用户也可以将其设置为'fast'，表示快速绘制，但这将会影响绘制的准确性。注意，该属性设置对使用 Z 缓冲算法和 OpenGL 算法的绘制操作没有任何影响。

线条、碎片、矩形和表面等图形对象都具有'EraseMode'属性，该属性用于指定它们如何被擦除和重绘。该属性的默认设置为'normal'，另外，用户也可以将其设为'none'，'background'和'xor'。当一个对象由于自身属性的改变或另外一个对象对其产生覆盖时，'EraseMode'属性将决定该对象的重画方式。比如，当'EraseMode'被设置为'none'时，若对象的属性被改变，则绘制修改后的对象时，不会擦除原来的对象。很明显，这会使坐标轴中留下了不希望出现的原来对象的影子。如果将'EraseMode'设置为'background'，则对象重绘时，会将原来对象的像素位置设置为坐标轴的背景颜色，不过，这样会破坏原来对象下层对象的外观。如果将'EraseMode'设置为'xor'，则对象重绘时，将通过对原来对象的像素和该点处下层对象的像素进行异或运算，得出最终重绘的像素值。该方式虽然不会破坏下层对象的表现，但是将会影响被改变对象的颜色。总的来说，将'EraseMode'设置为'none'可以最大限度地加快绘图速度，但会导致不希望出现的阴影；将'EraseMode'设置为'xor'或'background'则提供了绘图速度和绘图精度之间的折衷；将'EraseMode'设置'xor'则多用在生成动画的过程中。

31.16 回调

前面讲到，所有的句柄图形对象都具有'ButtonDownFcn'、'CreateFcn'和'DeleteFcn'属性。另外，图形对象还具有'CloseRequestFcn'、'KeyPressFcn'、'WindowButtonDownFcn'、'WindowButtonMotionFcn'以及'WindowButtonUpFcn'属性，用户界面函数则具有'CallBack'属性。上述这些属性有一个共同特点，就是与这些属性相关的属性值都是被称为“回调”的字符串。所谓回调，是指当与某一特定的属性相关联的用户动作发生时，需要执行的代码。在最简单的情况下，回调是在命令窗口工作区中由 eval 函数执行的字符串，并且该字符串可以由任何合法的 Matlab 语句构成。在大多数的情况下，回调都是函数调用，通常调用的就是定义回调的那个函数。用户也可以对回调进行设置，以便让 Matlab 执行特定的任务。回调是我们在下一章将要讲到的 Matlab 图形用户接口（GUI）的基础。

最简单的回调是关闭请求回调，在默认情况下，该回调不能为空，如下例所示：

```
>> get(gcf,'CloseRequestFcn')
ans =
closereq
>> class(ans)
ans =
char
```

默认情况下，当点击图形工具栏的关闭按钮关闭一个图形窗口时，字符串 closereq 就被传递给 eval 函数执行。该字符串是 Matlab 中的一个函数，仅仅用于删除当前图形窗口。因此点击关闭按钮就会删除相应的图形窗口。用户可以通过用另一个合法字符串替换

'CloseRequestFcn'，来改变上述的关闭请求操作的行为，例如：

```
>> set(gcf,'CloseRequestFcn','')
```

该语句用空字符串代替了'CloseRequestFcn'，因此就禁止了通过关闭按钮来实现关闭窗口的功能。set 函数的第二个字符串输入可以是任何合法的 Matlab 语句序列。通过该字符串的设置，用户可以在关闭窗口之前，获得确认关闭的提示信息。

31.17 M 文件示例

Matlab 本身就提供了众多的句柄图形的应用实例。在 specgraph 目录（用户可以在命令窗口中键入>>doc specgraph 来查看）中几乎所有的专用绘图函数都是由句柄图形函数构成的。即便是 M 文件函数 axis 也是通过调用句柄图形函数实现的。本节将通过几个例子对句柄图形的用法作更进一步的阐释。

下面给出的函数 mmis2d 用于判断一个坐标轴是否是定义于 x-y 平面的二维视图，如果是，就返回 True。

```
function [tf,xa,ya]=mmis2d(H)
%MMIS2D True for Axes that are 2D.
% MMIS2D(H) returns True if the axes having handle H displays
% a 2D viewpoint of the X-Y plane where the X- and Y-axes are
% parallel to the sides of the associated figure window.
%
% [TF,Xa,Ya]=MMIS2D(H) in addition returns the angles of x- and y-axes
%
% e.g., if the x-axis increases from right-to-left Xa=180
% e.g., if the y-axis increases from left-to-right Ya=0
% e.g., if the x-axis increases from bottom-to-top Xa=90

if ~ishandle(H)
    error('H Must be a Handle.')
end
if ~strcmp(get(H,'Type'),'axes')
    error('H Must be a Handle to an Axes Object.')
end
v=get(H,'view');
az=v(1);
el=v(2);
tf=rem(az,90)==0 && abs(el)==90;

if nargout==3
    xdir=strcmp(get(H,'Xdir'),'reverse');
    ydir=strcmp(get(H,'Ydir'),'reverse');
    s=sign(el);

    xa=mod(-s*az - xdir*180,360);
    ya=mod(s*(90-az) - ydir*180,360);
end
```

在 mmis2d 函数定义中，首先利用 ishandle 函数判断句柄 H 是否为合法的对象句柄，

若不是，就返回一条错误信息并终止程序；若是，便通过获取'Type'属性的值查看所提供的句柄是不是一个坐标轴句柄，如果不是，就返回一条错误信息并终止程序；如果是，再通过获取'View'属性决定所要求的输出。

我们再来看几个例子。下面所显示的函数 mmgetpos 用于找出并返回用特定单位表示的对象的位置向量。该函数首先获取当前的'Units'属性，然后将单位设置为期望输出的'Units'属性值，然后获取由设定的单位表示的对象'Position'属性值，最后将'Units'属性值恢复为原始值。

```
function p=mmgetpos(H,u,cp)
%MMGETPOS Get Object Position Vector in Specified Units.
% MMGETPOS(H,'Units') returns the position vector associated with the
% graphics object having handle H in the units specified by 'Units'.
% 'Units' is one of: 'pixels', 'normalized', 'points', 'inches', 'cent',
% or 'character'.
% 'Units' equal to 'data' is valid for text objects only.
%
% MMGETPOS does the "right thing", i.e., it: (1) saves the current units,
% (2) sets the units to those requested, (3) gets the position, then
% (4) restores the original units.
%
% MMGETPOS(H,'Units','CurrentPoint') returns the 'CurrentPoint' position
% of the figure having handle H in the units specified by 'Units'.
%
% MMGETPOS(H,'Units','Extent') returns the 'Extent' rectangle of the text
% object having handle H.
%
% 'Uimenu', 'Uicontextmenu', 'image', 'line', 'patch', 'surface',
% 'rectangle' and 'light' objects do NOT have position properties.

if~ischar(u)
    error('Units Must be a Valid String.')
end
if~ishandle(H)
    error('H is Not a Valid Handle.')
end
Htype=get(H,'Type');

if nargin==3 && ~isempty(cp) && ischar(cp)
       if strcmp(Htype,'figure') && lower(cp(1))=='c'
              pname='CurrentPoint';
       elseif strcmp(Htype,'text') && lower(cp(1))=='e'
              pname='Extent';
       else
              error('Unknown Input Syntax.')
       end
elseif H~=0
       pname='Position';
elseif H==0 % root object
       pname='ScreenSize';
```

```
else
      error('Unknown Input Syntax.')
end

hu=get(H,'units');       % get original units
set(H,'units',u);        % set to desired units

p=get(H,pname);          % get position in desired units
set(H,'units',hu)        % set units back to original units
```

下面所显示的mmzap函数使用了一项在编写句柄图形M文件函数时非常有用的技术：将 waitforbuttonpress 函数和 gco 函数联合起来使用来获取用鼠标选中对象的句柄。waitforbuttonpress函数是一个内嵌的Matlab函数，它等待一个鼠标点击或键盘按键按下的操作，其帮助文档如下：

```
>> help waitforbuttonpress
WAITFORBUTTONPRESS Wait for key/buttonpress over figure.
    T = WAITFORBUTTONPRESS stops program execution until a key or mouse
    button is pressed over a figure window. Returns 0 When terminated
    by a mouse buttonpress, or 1 when terminated by a keypress. Additional
    information about the terminating event is available from the current
    figure.
```

当鼠标指针位于图形之上时，按下鼠标后，gco 函数就会返回被选中的对象的句柄。后续函数就可以使用这个句柄对被选中的对象进行处理。mmzap函数的定义如下：

```
function mmzap(arg)
%MMZAP Delete Graphics Object Using Mouse.
% MMZAP waits for a mouse click on an object in
% a figure window and deletes the object.
% MMZAP or MMZAP text erases text objects.
% MMZAP axes erases axes objects.
% MMZAP line erases line objects.
% MMZAP surf erases surface objects.
% MMZAP patch erases patch objects.
%
% Clicking on an object other than the selected type,or striking
% a key on the keyboard aborts the command.

if margin<1
    arg='text';
end
Hf=get(0,'CurrentFigure');
if isempty(Hf)
   error('No Figure Window Exists.')
end
if length(findobj(0,'Type','figure'))==1
   figure(Hf) % bring only figure forward
end
key=waitforbuttonpress;
```

```
    if key  % key on keyboard pressed
          return
          else    % object selected
             object=gco;
             type=get(object,'Type');
             if strncmpi(type,arg,4)
                delete(object)
             end
    end
```

Matlab 提供了函数 xlim、ylim 和 zlim 允许用户分别设置和获取图形的 3 个坐标轴的坐标轴限。但刻度栅格线却不能针对一个坐标轴显示，当用户执行 grid 命令时，3 个坐标轴的刻度栅格线将同时显示和关闭。因此，为了能够实现针对某一坐标轴的刻度栅格线的显示和关闭，我们可以定义下面的函数 mmgrid：

```
function mmgrid(varargin)
%MMGRID  Individual Axis Grid Lines on the Current Axes.
% V is a single character X, Y, or Z denoting the desired axis.
% MMGRID V toggles the major grid lines on the V-axis.
% MMGRID V ON adds major grid lines to the V-axis.
% MMGRID V ON MINOR adds minor grid lines to the V-axis.
% MMGRID V OFF removes major grid lines from the V-axis.
% MMGRID V OFF MINOR removes minor grid lines from the V-axis.
%
% See also GRID

ni = nargin;
if ni==0
   error('At Least One Input Argument Required.')
end
if ~iscellstr(varargin)
   error('Input Arguments Must be Strings.')
end
Hf = get(0,'CurrentFigure'); % get current figure if it exists
if isempty(Hf) % no figure so do nothing
   return
end
Ha = get(Hf,'CurrentAxes'); % get current axes if it exists
if isempty(Ha) % no axes so do nothing
   return
end
% parse input and do work
V = varargin{1};
idx = strfind('xXyYzZ',V(1));
if isempty(idx)
   error('Unknown Axis Selected.')
end
VGrid = [upper(V(1)) 'Grid'];  % XGrid, YGrid, or ZGrid
```

```
if ni==1  % MMGRID V     Toggle Grid
   Gstate = get(Ha,VGrid);
   if strcmpi(Gstate,'on')
      set(Ha,VGrid,'off')
   else
      set(Ha,VGrid,'on')
   end
elseif ni==2  % MMGRID V ON  or MMGRID V OFF
   OnOff = varargin{2};
   if strcmpi(OnOff,'on')
      set(Ha,VGrid,'on')
   elseif strcmpi(OnOff,'off')
      set(Ha,VGrid,'off')
   else
      error('Second Argument Must be On or OFF.')
   end
elseif ni==3  % MMGRID V ON MINOR  or MMGRID V OFF MINOR
   if ~strcmpi(varargin{3},'minor')
      error('Unknown Third Argument.')
   end
   VGrid = [upper(V(1)) 'MinorGrid'];
   OnOff = varargin{2};
   if strcmpi(OnOff,'on')
      set(Ha,VGrid,'on')
   elseif strcmpi(OnOff,'off')
      set(Ha,VGrid,'off')
   else
      error('Second Argument Must be On or OFF.')
   end
end
```

上述函数定义中并没有使用 gcf 和 gca 函数获取当前图形和坐标轴对象的句柄，而是使用了句柄图形函数 get 获取当前对象的'CurrentFigure'和'CurrentAxes'属性。这样 Matlab 就不会自动创建图形和坐标轴，因此，如果当前不存在图形和坐标轴，那么 mmgrid 函数就不需要执行任何操作便可以终止。mmgrid 函数的后面部分根据输入参数对'XGrid'、'YGrid'、'ZGrid'、'XMinorGrid'、'YMinorGrid'或'ZMinorGrid'属性进行修改。

31.18 小结

句柄图形函数可以对 Matlab 中的可视化部件的外观进行精确调整。每个图形对象都有一个与之相对应的句柄，句柄图形函数都可以通过句柄来处理对象。下表列出了 Matlab 中的常用的句柄图形函数。

函数	描述
get	获取对象属性
set	设置对象属性
gcf	获取当前图形
gca	获取当前坐标轴
gco	获取当前对象
findobj	寻找含有指定属性的可见对象
findall	寻找含有指定属性的隐藏和可见对象
findfigs	将可见的图形窗口显示到计算机屏幕上
allchild	获得一个对象的隐藏和可见子对象的句柄
copyobj	将对象拷贝到一个新的父对象中
inspect	打开句柄图形属性检查框 GUI
root	计算机根对象，句柄值=0
figure	创建图形对象
axes	创建坐标轴对象
image	创建图像对象
light	创建光照对象
line	创建线条对象
patch	创建碎片对象
rectangle	创建矩形对象
surface	创建表面对象
text	创建文本对象
linkaxes	创建一个链接对象，将两个或多个坐标轴对象的坐标范围连接起来
linkprop	创建一个链接对象，将两个或多个句柄图形对象的属性列表连接起来
uibuttongroup	用于管理'radiobutton'和'togglebutton'类型控件对象的用户接口容器对象
uicontainer	创建用户接口容器对象
uicontrol	创建用户接口控件对象
uimenu	创建用户接口菜单对象
uicontextmenu	创建用户接口上下文菜单对象
uipanel	创建用户接口面板对象
uitoolbar	创建用户接口工具栏对象
uipushtool	创建瞬间接触按压按钮控件对象
uitoggletool	创建开关按钮控件对象
uitable	创建用户接口表格对象
uitree	创建用户接口树对象
uitreenode	创建用户接口树节点对象
areasseries	由函数 area 创建绘制对象

（续表）

函数	描述
barseries	由函数 bar、bar3 和 bar3h 创建绘制对象
contourgroup contourf	由函数 contour、contour3 创建绘制对象
errorbarseries	由函数 errorbar 创建绘制对象
lineseries	由函数 plot、plot3、semilogx、semilogy 和 loglog 创建绘制对象
quivergroup	由函数 quiver、quiver3 创建绘制对象
scattergroup	由函数 scatter、scatter3 创建绘制对象
stairseries	由函数 stairs 创建绘制对象
stemseries	由函数 stem、stem3 创建绘制对象
surfaceplot	由函数 surf、mesh 创建绘制对象
hggroup	创建组对象
hgtransform	创建组对象
makehgtform	创建 hgtransform 组对象所需的转换矩阵
annotation	创建注释坐标轴对象
reset	将对象属性重置为默认值
clf	清除当前图形
cla	清除当前坐标轴
ishandle	判断是否为对象句柄，若是，就返回 True，否则返回 False
delete	删除对象
close	使用关闭请求函数关闭图形
refresh	刷新图形
gcbo	获得当前回调对象
gcbf	获得当前回调图形
closereq	默认的图形'CloseRequestFcn'回调
newplot	根据已知的'NextPlot'属性的设置生成新的坐标轴

Chapter 32

图形用户接口

本章将向读者介绍 Matlab 提供的一类重要特性——图形用户接口。这些特性包括预定义的对话框、窗口菜单、上下文菜单、按钮、滚动条、单选按钮、切换按钮，弹出式菜单、列表框和工具栏，它们都可以用于用户创建的图形窗口中。除了上述特性以外，用户还可以动态跟踪鼠标的位置和运动，并根据这些位置和运动执行希望的任务。图形用户接口是一个非常广泛且详尽的功能，要将其介绍清楚可能需要一整本书的内容。因此，本书将通过几个典型的有价值的示例使读者能对图形用户接口特性有一个初步了解。

32.1 什么是图形用户接口（GUI）

所谓用户接口，是指用户和计算机之间或者用户和计算机程序之间进行通信的地点和实现交互的方法。总的来说，它是计算机和用户之间交换信息的方法。例如，计算机可以通过在计算机屏幕上显示文本和图形，或者利用扬声器发出声响向用户提供信息；用户可以利用键盘、鼠标、跟踪球、绘图板或者麦克风等输入设备向计算机提供信息。对于一台计算机、一个操作系统或应用程序而言，用户接口是一个十分重要的因素，它不仅定义了这些系统或程序提供给人的视觉外观和使用感觉，并且是用户在选择计算机或程序时的一个重要基础，因为舒服而高效的用户接口功能无疑会对用户产生极大的吸引力。

图形用户接口（GUI）是一个整合了诸如窗口、图标、按钮、菜单和文本等图形对象的用户接口。选中或者激活这些对象通常都会导致某个动作或变化的发生。最常用的激活方法是用鼠标或其他定点设备来控制屏幕上指针或光标的移动，并通过按下鼠标按键通知应用程序选中了一个对象或要执行其他的操作。

上一章我们讲到，句柄图形功能使用户可以自定义 Matlab 的信息显示方式，本章将要讲到的句柄图形用户接口函数同样可以使用户自定义其和计算机进行交互的方式。

本章通过讲述预定义对话框以及句柄图形 uicontrol 对象、uimenu 对象、uicontextmenu 对象和 uitoolbar 对象的使用方法，向读者展示如何在 Matlab 函数和 M 文件中添加图形用户接口。其中，uimenu 对象用于在图形窗口中生成下拉式菜单和其子菜单；uicontrol 对象用于生成诸如按钮、滑尺、弹出式菜单和文本框等类型的控件；uicontextmenu 对象用于生成在选中的对象上打开（通常利用右键）的上下文菜单；uitoolbar 对象则通常包含按钮（即

uipushtool 对象）和切换按钮（即 uitoggletool 对象）。

Matlab 提供了一个很好的范例用于演示其 GUI 功能。为了浏览这一范例，用户只需在命令窗口输入下面的命令就可以了：

```
>> demo
```

32.2 预定义对话框

从本质上讲，每一个计算机应用程序都会提供打开和保存数据文件的功能。为了查找文件打开或保存的路径，操作系统或窗口管理器会向用户提供一些对话框以便于用户指定文件名和正确的路径。在 Matlab 中，这些任务是由函数 uigetfile、uiputfile 和 uigetdir 完成的。需要注意的是，这 3 个函数本身并不执行文件打开和保存的操作，它们只返回用户需要使用的文件名或路径名，其函数内部使用了第 14 章讲到的底层文件 I/O 函数来执行文件和路径的操作。

在正常情况下，uigetfile、uiputfile 和 uigetdir 函数显示给用户的都是用户的操作系统平台支持的标准对话框。这些对话框的外观可能会因操作系统平台的不同而不同，但其功能都是一样的。实际上，这些对话框将使用与标准操作系统对话框相同的操作方式来指定用户需要打开或保存的文件名以及选择的路径。例如，如果用户想使用 uigetfile 函数寻找 startup.m 文件的位置，可以使用下面的命令（返回结果是作者的计算机得出的，读者在实验时可能有所不同）：

```
>> [fname,dirpath] = uigetfile ('*.m')
fname =
startup.m
dirpath =
C:\matlabR14\work\
```

如果用户需要得到一个包含文件名的完整路径信息，则可以使用下面的命令将文件名附加在路径名之后：

```
>> myfile = [dirpath fname]
myfile =
C:\matlabR14\work\startup.m
```

请读者注意：uigetfile 函数并不只局限于返回 Matlab 路径搜索列表中的路径，它可以返回任何在对话框关闭之前指定的路径。

如果用户想把一个 MAT 文件保存到当前 Matlab 路径（作者的当前路径为 Matlab 的 work 目录）中，可以使用 uiputfile 函数，代码如下：

```
>> [fname,dirpath] = uiputfile ('*.mat')
fname =
mydata
dirpath =
C:\matlabR14\work\
```

其中，mydata 是用户在保存对话框中输入的文件名。另外，尽管用户在命令中指明了文件名后缀为'*.mat'，但该后缀并不会自动附加到返回的 fname 后面，后缀名的添加需要用户来完成。

在 Matlab 中，使用 RGB 值指定颜色虽然简单易行，但却无法立刻看到所设置的颜色的视觉效果。另外，大部分图形对象都具有一个可以设置的'color'属性。因此，Matlab 提供了 uisetcolor 函数，用于显示选择 RGB 颜色值或设置某一图形对象的颜色属性时的用户图形接口。

uisetcolor 函数与 uigetfile、uiputfile 和 uigetdir 函数类似，也使用用户操作系统平台支持的标准颜色对话框。这些对话框的外观可能会因操作系统平台的不同而不同，但其功能都是一样的。

除了上述 GUI，还有一个内置的 GUI 函数比较有用，即 uisetfont。该函数用于显示选择字体和字体属性时的图形用户接口。例如，如果用户使用该函数选择了 12 点的 Palatino 斜体字体，将显示如下的结果：

```
>> fc = uisetfont
fc =
    FontName:       'Palatino'
    FontUnits:      'points'
    FontSize:       12
    FontWeight:     'normal'
    FontAngle:      'italic'
```

其中，fc 是包含了用户在 uisetfont 对话框中所选择的字体属性的结构体。该结果可以用于将任何文本对象的字体属性设置为它所规定的格式。

32.3 M 文件对话框

除了上一节讲到的预定义对话框外，Matlab 还提供了一些对话框用于完成 M 文件函数的功能。其中的一些对话框还提示用户输入某些参数。这些函数包括 axlimdlg、dialog、inputdlg、menu 和 msgbox。

axlimdlg 提供了一个用于设置坐标轴范围的基本对话框。该对话框很容易理解，这里不再赘述。inputdlg 函数将创建一系列文本编辑对话框（属于 uicontrol 对象），并提示用户在每一个对话框中输入数据。当程序需要用户输入多个值时，该函数将特别有用。Menu 函数向用户提供了使用按钮（属于 uicontrol 对象）时的多个选择。

dialog 和 msgdlg 函数用于创建最通用的对话框。其中，dialog 函数仅仅创建一个图形窗口，该窗口具有普通图形窗口的默认属性，只不过不包含任何坐标轴对象。使用 dialog 函数创建对话框将占用最少的系统资源。例如，它不为颜色映射分配内存。msgbox 函数用于生成各种消息对话框。当用户需要提供一个对话框并使之接收数据时，该函数可以用于管理底层的操作。

Matlab 中的标准消息对话框函数包括 errordlg、helpdlg、questdlg 和 warndlg。这些函数产生的对话框通常都包含一个特定的图标，另外还至少包含一个按钮用于确认和关闭对

话框。这些对话框不仅是用户进行程序开发时的有力助手，而且可以作为学习 GUI 构造时的典型示例。

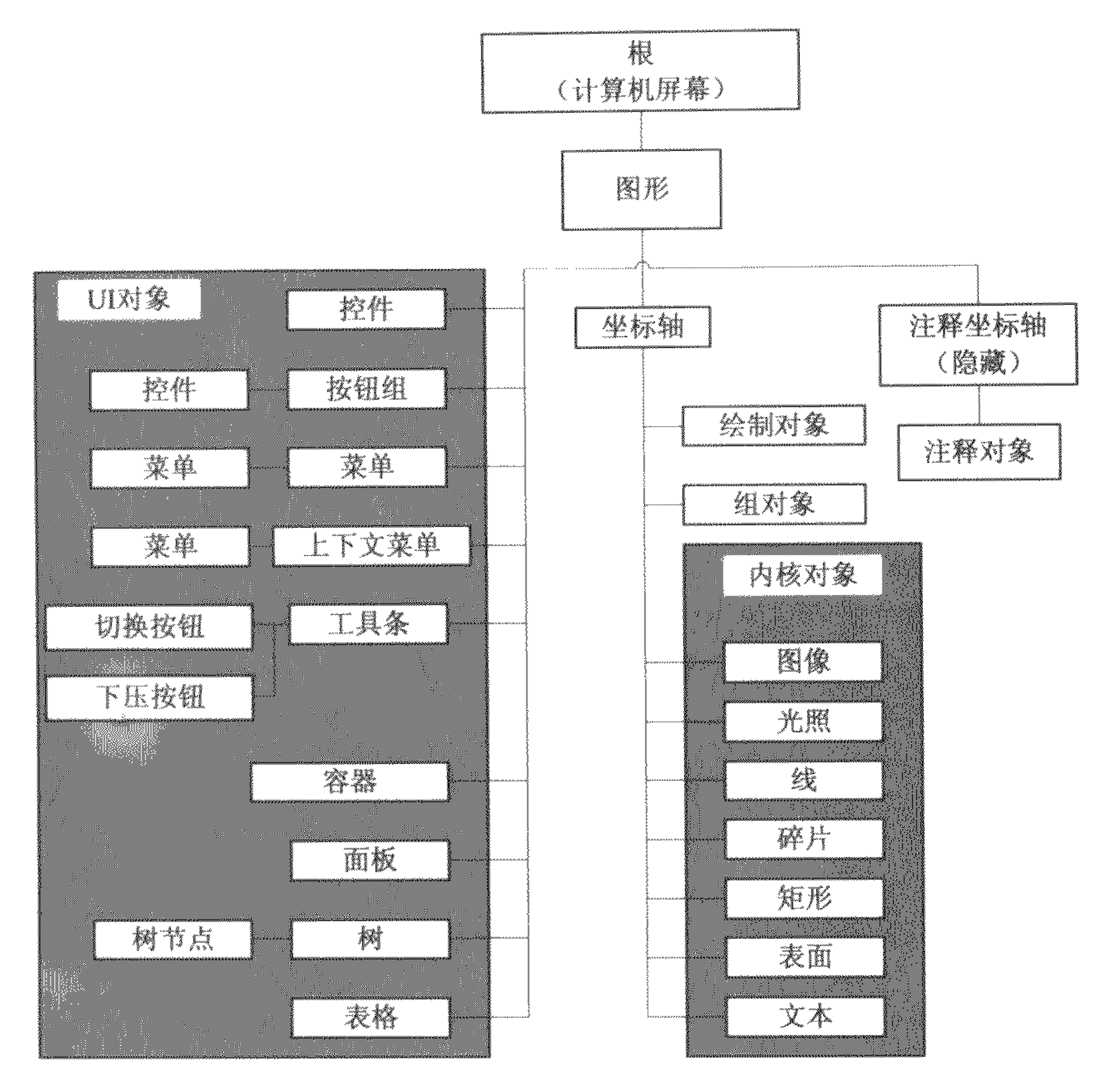
32.4　对话框小结

下表给出了 Matlab 提供的对话框函数的小结：

对话框函数	描述
axlimdlg	用于生成设置坐标轴范围的对话框
dialog	用于创建对话框或 GUI 类型的图形窗口
errordlg	用于生成错误消息对话框
helpdlg	用于生成帮助消息对话框
inputdlg	用于生成提示用户输入的对话框
listdlg	用于生成列表选择对话框
menu	用于生成菜单类型选择对话框
msgbox	用于生成通用消息对话框
pagedlg	用于生成页面位置对话框
pagesetupdlg	用于生成页面设置对话框
printdlg	用于生成打印对话框
printpreview	用于生成打印预览对话框
questdlg	用于生成询问对话框
uigetfile	用于生成标准的打开文件对话框
uiputfile	用于生成标准的保存文件对话框
uisetcolor	用于生成颜色选择对话框
uisetfont	用于生成字体和字体属性选择对话框
waitbar	用于显示等待进度条
warndlg	用于生成告警消息对话框

32.5　GUI 对象层次结构

实际上，Matlab 是利用一系列函数来创建图形用户接口的。这些函数主要用来创建用户接口（UI）类型的句柄图形对象，这一类图形接口对象已在上一章的对象层次结构图中进行了描述，为了使读者方便阅读，我们将该图重新绘制如下：



上图中的 UI 对象组包含众多类型的 GUI 对象，下面的表格对这些对象进行了总结：

UI 对象	描述
uicontrol	通用用户接口控制对象，有以下几种类型：'checkbox'（复选框）、'edit'（文本编辑框）、'frame'（组选框）、'listbox'（列表框）、'popupmenu'（下拉式菜单）、'pushbutton'（按钮）、'radiobutton'（单选按钮）、'slider'（滑动框）、'text'（静态文本框）、'togglebutton'（切换按钮）。以上所有的 uicontrol 对象都是图形对象的子对象
uimenu	用户接口菜单对象。当用来创建图形窗口顶部的菜单时，为图形对象的子对象；当用来创建菜单项或子菜单时，则为其他 uimenu 对象的子对象
uicontextmenu	用户接口上下文菜单对象，是图形对象的子对象。该对象的句柄通常出现在包含上下文菜单的句柄图形对象的'UIContextMenu'属性中。所有的 uicontextmenu 对象都包含一个或多个 uimenu 子对象用于创建其菜单项
uitoolbar	用户接口工具条对象，是图形对象的子对象。工具条对象通常包含一系列切换按钮（uitoggletool 对象）和按钮（uipushtool 对象）
uitoggletool	用户接口双向切换按钮，通常位于工具条上，是 uitoolbar 对象的子对象
uipushtool	用户接口瞬时下压按钮，通常位于工具条上，是 uitoolbar 对象的子对象
uitable	用户接口制表对象，是图形对象的子对象。一个制表对象就像一个电子表格一样具有多个行和列

（续表）

UI 对象	描述
uitree	用户接口树结构对象，是图形对象的子对象。树结构对象通常用于创建可视化的分级信息。例如，Matlab 的 Help 窗口中的 Help Navigator 区域内的 Content 表项给出的就是一个 Matlab 帮助信息的可视化分级树状图。另外，用户通常在 Windows 的资源管理器中看到的文件夹及其子文件夹的树状结构也是 uitree 对象的一个实例
uitreenode	用户接口树节点对象，通常用来定义一个 uitree 对象中的的节点
uibuttongroup	用户接口容器对象，是图形对象的子对象。通常以 uicontrol 对象以及其他句柄图形对象为其子对象。uibuttongroup 对象的主要作用是管理其所包含的排他性选择按钮（包括单选按钮和切换按钮）的行为
uicontainer	用户接口容器对象，通常包含一些其他的用户接口对象并成为这些对象的父对象。uicontainer 对象中的子对象的位置和移动属性都是相对于该容器对象而言的
uipanel	用户接口容器对象。与 uicontainer 对象相似，该对象也包含一些其他的用户接口对象并成为这些对象的父对象。不过，uipanel 对象具有可以设置的边界和标题

从上面的表格可以看出，uicontrol 对象包含 10 种不同的类型，每种类型都用于创建特定的 GUI 对象。这些类型都是创建图形用户接口的基础构件，因此，下表对它们进行了总结：

uicontrol 类型	描述
'checkbox'	用于创建复选框，一个复选框通常包含多个复选项，每个复选项由一个方形的选择框和一个紧随其后的标签组成。当一个复选项被激活后，该项将会在选中和清除之间切换。如果一个复选项被选中，则前面的方形选择框内会出现一个'×'号，当该复选项被清除后，'×'号就会消失。复选框通常用于进行状态或属性的选择
'edit'	用于创建可编辑的文本框。用户可以动态地修改或替换文本框中的内容，就像使用一个文本编辑器或字处理器一样。该文本框中可以包含一行或多行文本（需要用户在其属性列表中设置是显示一行文本还是多行文本）。如果用户设置了单行可编辑文本框，则该文本框只能接受一行用户输入；如果用户设置了多行可编辑文本框，则该文本框可以接受多行用户输入。在多行输入时，如果输入的文本超出了文本框的范围，则该文本框会自动生成一个纵向的滚动条。无论单行还是多行文本框，当用户键入回车键时，都表明输入结束；对于多行文本框，用户换行需要同时键入 ctrl 键和回车键
'frame'	用于生成组选框，组选框是一个透明的、带边框和阴影的矩形区域。组选框与 uimenu 对象的'Separator'属性类似，都提供了可视化的分割。组选框通常用于对单选按钮或其他 uicontrol 对象进行逻辑分组。组选框必须先于其中的 uicontrol 对象建立，否则它将会覆盖这些 uicontrol 对象。在 Matlab 7 中，通常用 uicontainer 和 uipanel 对象来代替组选框类型的 uicontrol 对象，因为这些对象用途更加广泛
'listbox'	用于生成列表框，列表框看起来与多行文本框类似。只不过其中的文本只能用于选择，不能进行编辑

（续表）

uicontrol 类型	描述
'popupmenu'	用于生成下拉式菜单。下拉式菜单向用户提供了多个选项，但用户只能从其中选择一项。当关闭该菜单时，它将会变成一个包含当前用户选择项的矩形或按钮，并位于一个凸起的矩形区域内。在用户选择项右侧还有一个向下的箭头，当用户用鼠标单击该箭头时，就会弹出其他各个选项。如果用户要选择一个新的选项，只需上下移动鼠标，并在希望的选项上单击，就可以完成新的选择。通常 MS Windows 和其他一些基于窗口的操作系统中都会向用户提供下拉式菜单，以便用户在多个选项中做出选择
'pushbutton'	用户创建按钮，有时也称为命令按钮。按钮是一个比较小的矩形对象，通常上面还有一个文本标签。按钮是用户最常见的图形对象，用户可以将鼠标移动到按钮上，然后单击，就可以使 Matlab 执行定义在该按钮上的回调函数。按钮被按下去后，会立即恢复到原始的状态（弹起状态）。按钮通常用于执行某一个操作，而不是用于进行状态或属性的选择
'radiobutton'	用于创建单选按钮。单选按钮是一个由标签和一个很小的圆圈或菱形组成的控件。当被选中时，圆圈和菱形会呈现填充的状态，取消选中时，又会恢复到无填充状态。单选按钮通常用于从一组选择项中选择单个选项
'slider'	用于创建滑动框。滑动框又称为滚动条，通常由 3 部分构成：滑杆（代表有效对象值范围的一个矩形区域），指示器（位于滑杆内部代表当前值的一个指示标志），位于滑杆两端的箭头。滑动框通常用于从一个数据范围选择一个数据值。在滑动框中选择数值的方法有 3 种：①按住鼠标左键，移动鼠标使鼠标光标在滑动框内移动，当移动到希望的数据位置时，放开鼠标左键，即可选中该位置上的值。②当鼠标位于滑杆内时，连续单击鼠标，使指示器朝一个方向移动，用户每单击一下鼠标，指示器大致移动 1/10 滑杆长度，直到用户得到需要的数值为止。③将鼠标放在滑杆两端的其中一个箭头上，然后单击鼠标，用户每单击一次，指示器大致移动 1/100 滑杆长度，直到用户得到需要的数值为止。滑动框通常会和其他一些文本对象一起使用，用来表示该滑动框的标题、当前值和滑动范围
'text'	用户创建静态文本框。静态文本框是用于显示文本字符串的控件，通常用于显示标题、标签、用户信息或当前值。其中的静态属性意味着用户无法对其进行动态操作（能进行动态操作的文本框为'checkbox'）
'togglebutton'	用户创建切换按钮。切换按钮与普通按钮一样，只不过当鼠标单击一次切换按钮时，会交替呈现两种不同的状态（弹起和按下）（有点类似于电源开关），而普通的按钮会在鼠标单击后立即弹起。切换按钮和普通按钮一样，鼠标每单击一次就执行一次相应的操作

根据前面两个表格给出的 12 种 GUI 对象和 10 类 uicontrol 对象，用户在创建图形用户接口时一共可以使用 21 种完全不同的 GUI 对象。其中每一种 GUI 对象都具有多个句柄图形属性，这些属性也都可以利用 set 和 get 函数进行修改和访问。因此，要将所有这些 GUI 对象、对象的属性及属性值都介绍清楚，显然不是三言两语就能完成的。限于篇幅原因，本书后面的章节只对 GUI 创建过程中的一些主要特性进行简介，并在最后给出几个典型的示例。

32.6　GUI 创建的基本步骤

GUI 创建的基本步骤为：

（1）首先弄清希望 GUI 进行什么样的操作。这是最重要也是最难的一步。很多情况下，在用户创建 GUI 的过程中还要涌现一些新的想法或发现一些新的问题，用户需要重新回到这一步进行思考。

（2）在纸上画出 GUI 对象的大致布局。大多数用户可能会跳过这一步。但从长远角度讲，这一步可以节省用户的时间，因为在纸上反复勾画可能的 GUI 布局要比直接在 Matlab 中创建和修改来的更快（尤其是比较复杂的布局）。

（3）根据最终勾画的布局，使用适当的 UI 对象创建 GUI。这一步既可以通过直接在一个 M 文件中输入相应的代码实现，也可以使用 Matlab 提供的图形用户接口开发环境（GUIDE）来实现。

（4）创建需要与用户进行交互的回调代码（例如单击一个按钮时需要执行的代码）。回调代码通常与创建 GUI 的代码存在同一个文件中。

（5）验证和调试 GUI。尤其当该 GUI 可能会被别的用户使用时，就需要从一个不熟悉该 GUI 的用户角度出发（而不是从开发者的角度出发），反复与该 GUI 进行各种方式的交互。

32.7　GUI 对象的大小和位置

UI 对象的'Position'和'Units'属性通常被用来设定其相对于父对象（通常是图形对象）的相对位置。如果给定父对象（图形对象或其他容器对象）的大小，则其中的 GUI 对象的布局设计其实就是简单的几何问题。在设计一个 GUI 时，用户必须保证该对象具有足够的尺寸，以便能够显示其中的文本或图像，另外，用户还需要考虑该对象是否具有可以改变的尺寸。

通常，UI 对象的大小和布局设置是一个不断尝试和调整的过程。即便是用户已经调整满意的结果，移植到另一个操作系统平台上也可能会出现很大的不同，用户仍需要进一步调整。大部分情况下，建议用户使 UI 对象比需要的外观稍微大一些，这样可以使该对象在所有的平台上都能够使用。

虽然一个图形有默认的大小，但是并不能保证所有的图形在所有的平台上都是那个大小。如果用户在一个现有的图形中添加了 UI 对象，则可能会发现该图形比默认的大小要小或要大一些。另外，用户可以随时改变图形的大小，除非该图形的'Resize'属性设置为'off'。因此，当用户需要在一个图形窗口中创建 GUI 时，重新设置该图形窗口的大小是很必要的。

当用户在一个可改变大小的图形窗口中添加 UI 对象时，需要格外注意'Units'属性和固定字体大小的字符串所带来的限制。当各个 UI 对象的位置是用诸如像素、英寸、厘米或者点这样的绝对单位指定的时候，这些对象的大小和位置就不会因图形窗口大小的改变而改变。由于这些对象保持的是相对于图形窗口左下角的绝对位置，因此，如果图形窗口变小，

那么某些 UI 对象就会位于图形窗口之外，从而变得不可见。

当 UI 对象的位置是用标准单位指定时，这些对象保持的是相互之间以及与图形窗口之间的相对位置关系，因此，当图形窗口的大小发生变化时，这些对象的大小和位置也会发生变化，以保证上述相对关系保持不变。然而，这种表示方法有一个明显的缺陷，就是如果图形窗口变得很小，就会导致 UI 对象变得太小，以至于其中的标签字符串不能完整显示，从而失去可读性（这是因为字符串的字体大小是固定不变的，而超出了 UI 对象尺寸的那部分字符都将被剪切掉）。

上述字符串的固定大小和 GUI 的可变大小之间的矛盾通常是难以避免的。一个比较好的方法是在 UI 对象周围或其内部留出足够的空白区域，以便当 GUI 从一个系统平台移植到另一个系统平台时不会发生标签字符的剪切。如果一个 GUI 是大小可变的，那么用户就可以使用该 GUI 图形的'ResizeFcn'回调属性来移动 UI 对象或重新设置 UI 对象的形状，以便响应图形窗口大小的改变。

32.8 捕获鼠标动作

GUI 函数通常利用鼠标指针的位置和鼠标按键的状态来控制 Matlab 的动作。本节主要讨论鼠标指针和对象位置与鼠标按键动作之间的交互，以及 Matlab 如何对变化或事件（例如按下鼠标键，释放鼠标键或鼠标指针移动）作出响应。

所有的句柄图形对象都有一个'ButtonDownFcn'属性。大部分 UI 对象都有一个'CallBack'属性，其中 uicontrol 对象还有一个'KeyPressFcn'属性。图形对象拥有'WindowButtonDownFcn'、'WindowButtonUpFcn'、'WindowButtonMotionFcn'、'KeyPressFcn'、'CloseRequestFcn'以及'ResizeFcn'属性。另外，所有的图形对象还有'CreateFcn'和'DeleteFcn'属性。上述属性的值都是一个回调字符串，当这些属性被激活时，Matlab 就会执行回调字符串所代表的回调代码。鼠标指针的位置决定了当事件发生时执行哪些回调，以及这些回调被激活的顺序。

前一章已经讨论了与本节有关的层叠顺序和对象的有效选择区域问题。Matlab 通常根据图形中的 3 个区域来决定哪个回调被激活：①当鼠标指针位于由属性'Position'确定的句柄图形对象范围之内时，就认为鼠标指针位于该对象之上。②如果鼠标指针没有位于对象之上，但位于对象的有效选择区域内，就认为鼠标指针位于该对象附近。③如果鼠标指针既不在一个对象之上，也不在一个对象附近，就认为鼠标指针远离该对象。当对象或者它们的选择区域出现相互重叠时，通常利用叠放次序决定哪一个对象被鼠标选中。

关于线条、表面、碎片、文本和坐标轴等句柄图形对象的选择区域已在前一章进行了讨论。uimenu 对象没有扩展的有效选择区域，也就是说，要么鼠标指针位于 uimenu 对象之上，要么远离该对象。uicontrol 对象则具有有效选择区域，该区域是控件的边界向外扩展 5 个像素的范围，因此，鼠标指针既可以位于 uicontrol 对象之上，也可以位于该对象附近。

鼠标单击被定义为当鼠标指针位于一个对象上方或附近时，按下鼠标键然后立刻释放该键的过程。如果鼠标指针位于一个 uimenu、uicontextmenu 或 uicontrol 对象之上，只要对象的'Enable'属性被设置为'on'，那么鼠标单击就会触发对象的'CallBack'属性字符串执行。一

般而言，按下鼠标键将使 uicontrol 对象做好被触发的准备，并且通常会改变 uicontrol 或 uimenu 的外观（例如呈凹陷的效果），而释放鼠标键就触发了回调的执行。如果鼠标指针不在一个 uicontrol 或 uimenu 对象之上，那么鼠标键按下和释放动作将按照下面解释的程序触发。

当鼠标键被按下时，就生成了一个鼠标键按下事件。当该事件发生并且鼠标指针位于图形窗口内部时，将会发生多个不同的动作，具体是什么动作要取决于鼠标指针的位置和该指针与哪个句柄图形对象接近。如果鼠标选中了某个对象，则该对象就成为当前对象。如果没有选中对象，那么就将图形本身视为当前对象，这时，图形的'CurrentPoint'和'SelectionType'属性也将被更新。当鼠标键被释放时，就会触发相应的回调执行。

下表列出了不同鼠标指针位置时，鼠标键按下事件所触发的回调响应或采取的动作：

鼠标键按下时鼠标指针的位置	所采取的动作
鼠标指针位于 uimenu 菜单项之上，且菜单对象的'Enable'属性为'on'	改变 uimenu 对象的外观并为释放鼠标键做好准备
鼠标指针位于 uicontrol 对象之上，且该对象的'Enable'属性为'on'	改变 uicontrol 对象的外观并为释放鼠标键做好准备
鼠标指针位于 uimenu 菜单项之上，且菜单对象的'Enable'属性为'off'	忽略鼠标键按下事件
鼠标指针位于 uicontrol 对象之上，且该对象的'Enable'属性为'off'或'inactive'	首先执行图形对象的'WindowButtonDownFcn'回调代码，然后执行 uicontrol 对象的'ButtonDownFcn'回调代码
鼠标指针位于除了 uimenu 和 uicontrol 之外的其他句柄图形对象之上或附近	首先执行图形对象的'WindowButtonDownFcn'回调代码，然后执行鼠标所指对象的'ButtonDownFcn'回调代码
鼠标指针位于图形对象内部，但不在其他任何对象之上或附近	首先执行图形对象的'WindowButtonDownFcn'回调代码，然后执行其'ButtonDownFcn'回调代码

由上表可知，当鼠标指针位于除 uicontrol 和 uimenu 对象之外的其他对象之上时，鼠标键按下事件总会在触发选中对象的'ButtonDownFcn'回调代码之前触发图形对象的'WindowButtonDownFcn'回调代码。当鼠标指针位于 uicontrol 对象附近或位于'Enable'属性为'off'或'inactive'的 uicontrol 对象之上时，鼠标按下事件首先完成图形的'WindowButtonDownFcn'回调，然后执行 uicontrol 对象的'ButtonDownFcn'回调（注意不是'CallBack'回调）。uimenu 对象的'ButtonDownFcn'回调永远不会被触发。

当鼠标键被释放时，就产生一个鼠标键释放事件。当该事件发生时，图形对象的'CurrentPoint'属性首先被更新，同时图形对象的'WindowButtonUpFcn'回调被触发。如果'WindowButtonUpFcn'回调没有定义，那么在鼠标键释放事件就不会更新图形对象的'CurrentPoint'属性。

当鼠标指针被移动到图形内部时，就会产生鼠标指针移动事件。当该事件发生时，图形的'CurrentPoint'属性将被更新，同时图形的'WindowButtonMotionFcn'回调被触发。如果

'WindowButtonMotionFcn'回调没有定义，那么当鼠标指针移动时，'CurrentPoint'属性也不会被更新。

32.9 事件队列

由于 GUI 完全依赖于用户的动作，因此 GUI 的执行是以事件队列的概念为基础的。也就是说，当 GUI 展现在用户面前后，用户就可以以任意方式和 GUI 进行交互，并且每一次交互都会生成一个事件，该事件将被按顺序放入事件队列中。除此以外，其他涉及到利用图形窗口输入或输出的命令也会产生事件。事件既包括鼠标指针的移动和触发回调执行的鼠标按键的按下与释放，又包括 waitfor、waitforbuttonpress 函数，以及 drawnow、figure、getframe 或 pause 等图形重绘函数。对于所有这些事件，Matlab 都按照它们出现在事件队列中的顺序进行操作。在大多数情况下，Matlab 通常利用默认方式管理事件队列（即按顺序执行），但大部分句柄图形对象都具有'Interruptible'和'BusyAction'属性，用于声明特殊的动作或优先执行的操作。

一般情况下，一个回调执行是不能被中断的，除非它遇到一个 waitfor、drawnow、waitforbuttonpress、getframe、pause 或 figure 命令。当回调执行到这些特殊命令时，Matlab 就会将回调执行“挂起”，并检测事件队列中的每个“未完成”事件。如果与回调相对应的对象的'Interruptible'属性被设置为'on'（这是这个属性的默认值），那么 Matlab 就将所有未完成的事件处理完毕，然后再恢复这个被“挂起”的回调；如果'Interruptible'属性被设置为'off'，那么 Matlab 只处理没有完成的屏幕刷新事件。另外，如果对象的'BusyAction'属性被设置为'cancel'，则中断回调执行的事件就被忽略；如果'BusyAction'属性被设置为'queue'，那么中断回调执行的事件将被保留在事件队列中，直到被中断的回调执行完毕后再响应该事件。即使一个回调执行不能被中断，当回调执行遇到一个 waitfor、waitforbuttonpress、drawnow、figure、getframe 或 pause 命令时，未完成的屏幕更新事件也会被执行。

32.10 回调编程

句柄图形和 GUI 函数都充分利用回调来扩展执行用户选择的任务。在 Matlab 7 之前的版本中，回调都是可以在命令窗口执行的字符串。此时，激活这些回调需要将回调字符串传递给 eval 函数。例如，下面的代码将按钮对象的回调函数设置为字符串'myguifcn push1'：

```
H_p1 = uicontrol('Style','PushButton',...
                 'Callback','myguifcn push1');
```

当该 GUI 中的按钮被按下时，上述语句中的回调语法将被解释为一个可以在命令窗口执行的函数调用，其形式为：myguifcn('push1')。在大部分情况下，myguifcn 是一个包含了上述 uicontrol 语句的函数。也就是说，回调函数通常都直接调用定义它的函数。这样的话，我们就可以使用一个 M 文件创建 GUI，并定义当 GUI 运行时的回调执行代码。将所有的 GUI 回调集中在一个单独的文件中，需要用到切换编程结构（switchyard programming structure）。例如，假如一个 GUI M 文件函数中创建了 3 个按钮——Apply、Revert 和 Done，

它们的回调函数分别为 myguifcn('Apply')、myguifcn('Revert')和 myguifcn('Done')。那么 myguifcn 函数就可以使用 Switch-Case 语句来实现切换编程回调结构。下面给出了 myguifcn 函数的具体内容：

```
function myguifcn(arg)
%MYGUIFCN Sample Switchyard Programming Example.

persistent mydata
if nargin==0
    arg= 'Initialize';
end
switch arg
case 'Initialize'
   % code that creates the GUI and sets the callbacks
   uicontrol('Style','PushButton',...
             'String','Apply',...
             % In a real GUI other properties must be set also
             'Callback','myguifcn Apply')
   uicontrol('Style','PushButton',...
             'String','Revert',...
             % In a real GUI other properties must be set also
             'Callback','myguifcn Revert')
   uicontrol('Style','PushButton',...
             'String','Done',...
             % In a real GUI other properties must be set also
             'Callback','myguifcn Done')
   mydata = ... % data to be stored in the GUI for callback use,
                % most commonly a structure

case 'Apply'
   % code that performs Apply button callback actions

case 'Revert'
   % code that performs Revert button callback actions

case 'Done'
   % code that performs Done button callback actions

otherwise
   % report error?
end
```

从上面的函数定义可以看出，回调编程的难点在于获取数据。尽管切换编程回调每次调用的都是同一个函数，但每一次回调都创建一个独立的函数实例，并且具有独立的函数工作区。因此，回调实例除了访问通过回调字符串传递给函数的参数外，不能访问其他任何参数或数据。为此，Matlab 提供了几种方式使数据能够被回调实例访问。最简单的方法是将数据变量声明为 persistent 类型，因为声明为 persistent 的变量将会在整个 Matlab 执行周期保持存在，函数可以随时对其进行访问。在前面的函数体中，变量 mydata 被声明为 persistent，并在创建 GUI 时对其进行赋值，这样的话，当切换回调被调用时，存储在 mydata

中的数据就可以随时被访问。使用 persistent 声明的一个缺陷是在每一时刻只能有一个 GUI 实例存在。如果有多个 GUI 同时存在，它们将共享同一个 persistent 数据。

除了使用 persistent 变量外，句柄图形对象的'tag'属性也可以被设置为一个惟一的字符串。这样，用户就可以使用 findobj 函数来寻找具有指定 tag 字符串属性的对象。找到这些对象后，getappdata、setappdata、rmappdata 和 isappdata 函数就可以用来对这些对象中的数据进行处理。下面的函数 myguifcn 就是采用这种方法的回调函数。

```
function myguifcn(arg)
%MYGUIFCN Sample Switchyard Programming Example.

if nargin==0
     arg= 'Initialize';
end
switch arg
case  'Initialize'
    % code that creates the GUI and sets the callbacks
    Hf = figure(...   %Create figure for gui
                % In a real GUI other properties must be set also
                'Tag','MyGuiFcnTag')
    uicontrol('Parent',Hf,...
              'Style','PushButton',...
              'String','Apply',...
              % In a real GUI other properties must be set also
              'Callback','myguifcn Apply')
    uicontrol('Parent',Hf,...
              'Style','PushButton',...
              'String','Revert',...
              % In a real GUI other properties must be set also
              'Callback','myguifcn Revert')
    uicontrol('Parent',Hf,...
              'Style','PushButton',...
              'String','Done',...
              % In a real GUI other properties must be set also
              'Callback','myguifcn Done')
    mydata = ...  % data to be stored in the GUI for callback use,
                % most commonly a structure
    setappdata(Hf,'MyGuiFcnData',mydata)

case 'Apply'
    % code that performs Apply button callback actions
    Hf = findobj('Tag','MyGuiFcnTag');
    mydata = getappdata(Hf,'MyGuiFcnData');

case 'Revert'
    % code that performs Revert putton callback actions
    Hf = findobj('Tag','MyGuiFcnTag');
    mydata = getappdata(Hf,'MyGuiFcnData');

case  'Done'
    % code that performs Done button callback actions
```

```
    Hf = findobj('Tag','MyGuiFcnTag');
    mydata = getappdata(Hf,'MyGuiFcnData');

otherwise
    % report error?
end
```

在上面的函数定义中，setappdata 函数用于在 GUI 图形的应用数据属性中保存 mydata 变量。GUI 的标签属性被指定为'MyGuiFcnTag'。在每次回调中，findobj 函数用于查找 GUI 图形的句柄，getappdata 函数则用来获取图形中保存的应用数据。

另外，用户还可以使用 guidata 函数将数据保存在 GUI 图形的'ApplicationData'属性中，或从 GUI 图形的'ApplicationData'属性中读取数据。因此，上面的函数所实现的功能同样可以利用 guidata 函数在下面的函数中实现。

```
function myguifcn(arg)
%MYGUIFCN Sample Switchyard Programming Example.

if nargin==0
    arg= 'Initialize';
end
switch arg
case  'Initialize'
   % code that creates the GUI and sets the callbacks
   Hf = figure(...)   %Create figure for gui

   uicontrol('Parent',Hf,...
             'Style','PushButton',...
             'String','Apply',...
             % In a real GUI other properties must be set also
             'Callback','myguifcn Apply')
   uicontrol('Parent',Hf,...
             'Style','PushButton',...
             'String','Revert',...
             % In a real GUI other properties must be set also
             'Callback','myguifcn Revert')
   uicontrol('Parent',Hf,...
             'Style','PushButton',...
             'String','Done',...
             % In a real GUI other properties must be set also
             'Callback','myguifcn Done')
   mydata = ...  % data to be stored in the GUI for callback use,
                 % most commonly a structure
   guidata(Hf,mydata)

case 'Apply'
   % code that performs Apply button callback actions
   mydata = guidata(gcbo);

case 'Revert'
   % code that performs Revert button callback actions
```

```
        mydata = guidata(gcbo);

    case  'Done'
        % code that performs Done button callback actions
        mydata = guidata(gcbo);

    otherwise
        % report error?
    end
```

在上面的函数中，不再需要为对象添加标签，但需要使用 gcbo 函数获取回调对象的句柄。也就是说，gcbo 函数将返回调用回调的对象的句柄。如果该句柄不是包含数据的图形的句柄，guidata 函数将会在回调对象的父对象中寻找图形。实际上，guidata 函数在后台也使用了 setappdata 和 getappdata 函数，并简化了在回调实例之间存储和获取数据的语法。

自从引入句柄图形函数之初，Matlab 就已经将回调定义为可以在命令窗口的工作区中执行的字符串。其实，这种方法也有许多弊病，例如回调字符串只能在每次调用回调的时候被执行，数据不能直接传递给回调等。为了克服这些弊病，Matlab 7 增加了一些新的定义回调的方法。比如，Matlab 7 中允许使用函数句柄来定义回调，例如，下面的代码将 uicontrol 对象的回调定义为函数 mycbfcn 的句柄：

```
H_pl = uicontrol('Style','PushButton',...
                 'Callback',@mycbfcn);
```

当上述回调被调用时，Matlab 将通过执行 mycbfcn(cbo,eventdata)调用指向回调的函数。其中第一个参数 cbo 包含了回调对象的句柄（即函数 gcbo 的返回值），第二个参数包含了指定给回调对象的事件数据和回调的类型。在上面的代码中，变量的内容并没有被指定。

采用函数句柄的方法定义回调的一个重要好处是在函数句柄被创建（也即 GUI 被创建）时，执行该函数所需要的所有信息也都被同时创建了。也就是说，通过@mycbfcn 定义的指向回调的函数也可以作为创建 GUI 的 M 文件函数的子函数或内嵌函数。如果采用函数句柄方法，上面的回调函数也可以写成下面的函数形式。

```
function myguifcn
%MYGUIFCN Sample GUI Function Using Function Callbacks.

% code that creates the GUI and sets the callbacks
Hf = figure(...)   %Create figure for gui

uicontrol('Parent',Hf,...
          'Style','PushButton',...
          'String','Apply',...
          % In a real GUI other properties must be set also
          'Callback',@Applyfcn)
uicontrol('Parent',Hf,...
          'Style','PushButton',...
          'String','Revert',...
          % In a real GUI other properties must be set also
          'Callback',@Revertfcn)
```

```
uicontrol('Parent',Hf,...
          'Style','PushButton',...
          'String','Done',...
          % In a real GUI other properties must be set also
          'Callback',@Donefcn)
mydata = ...  % data to be stored in the GUI for callback use,
              % most commonly a structure
guidata(Hf,mydata)

%--------------------------------
function Applyfcn(cbo,eventdata)
% subfunction code that performs Apply button callback actions
mydata = guidata(cbo);

%--------------------------------
function Revertfcn(cbo,eventdata)
% subfunction code that performs Revert button callback actions
mydata = guidata(cbo);

%--------------------------------
function Donefcn(cbo,eventdata)
% subfunction code that performs Done button callback actions
mydata = guidata(cbo);
```

上面的函数不再需要使用切换编程，每一个回调调用的都是自己的子函数。另外，由于回调对象句柄是函数 guidata 的第一个参数，因此获取 GUI 图形中的存储数据更加方便。

除了将函数句柄用作回调这种默认方式以外，Matlab 还支持以多个数据变量为参数的函数句柄。例如，下面的代码将 uicontrol 对象的回调定义为一个由函数句柄@mycbfcn、附加输入参数 mydata1 和 mydata2 构成的单元数组。当该回调被调用时，Matlab 将执行下面的命令：

```
mycbfcn(cbo,eventdata,mydata1,mydata2)
```

其中，变量 mydata1 和 mydata2 的值将在 GUI 创建时设置。另外，回调函数可以支持任意数量的附加变量参数。

使用附加变量参数，上面的回调函数定义也可以用下面的函数完成：

```
function myguifcn
%MYGUIFCN Sample GUI Funcition Using Function Callbacks.

% code that creates the GUI and sets the callbacks
Hf = figure(...)   %Create figure for gui

Mydata = ...  % Data to be stored in the GUI for callback use,
              % most commonly a structure

uicontrol('Parent',Hf,...
          'Style','PushButton',...
          'String','Apply',...
          % In a real GUI other properties must be set also
```

```
            'Callback',{@Applyfcn,mydata})
uicontrol('Parent',Hf,...
            'Style','PushButton',...
            'String','Revert',...
             % In a real GUI other properties must be set also
            'Callback',{@Revertfcn,mydata})
uicontrol('Parent',Hf, ...
            'Style','PushButton',...
            'String','Done',...
            % In a real GUI other properties must be set also
            'Callback',{@Donefcn,mydata})
%----------------------------------------
function Applyfcn(cbo,eventdata,mydata)
% subfunction code that performs Apply button callback actions
%----------------------------------------
function Revertfcn(cbo,eventdata,mydata)
% subfunction code that performs Revert button callback actions
%----------------------------------------
function Donefcn(cbo,eventdata,mydata)
% subfunction code that performs Done button callback actions
```

在上面的函数中，不需要使用 guidata 函数来获取数据。不过，变量 mydata 的内容必须在回调单元数组出现之前给出。

尽管上面的例子都将子函数的函数句柄作为回调，但实际上，作为回调的函数句柄可以是任何类型的函数句柄，只要该函数句柄在回调创建时可见，即位于回调执行周期内。因此，除了子函数外，内嵌函数、匿名函数或私有函数都可以作为函数句柄回调。一旦函数句柄被创建，哪怕该函数超出了 Matlab 的可视化范围之外，一样能被 Matlab 执行。可见，功能强大的函数句柄被用于回调，将使得 Matlab 的 GUI 编程效率大大提高。

32.11 M 文件示例

利用非常准确的例子来展示 GUI 编程并不是一件容易的事情，这是因为 GUI M 函数文件一般是非常长的，并且它们的用户界面特性也不可能在一张静态的打印纸上完全展现出来。因此，熟悉 GUI 编程的一种最好最简单的方法是在 Matlab 或网页中寻找一个现成的 GUI，并用一个适当的编辑器查看它的源代码，然后试着运行，观察这些代码产生什么样的执行效果。将 GUI 运行的可视化结果和生成这个结果的 Matlab 代码进行比较，我们就可以得到大量的信息，而这些信息是无法通过其他途径得到的。

根据上面的方法，本节展示了两个 GUI 实例。在第一个实例中，函数 mmtext6 和 mmtext7 演示了图形对象的'WindowButtonDownFcn'、'WindowButtonMotionFcn'和'WindowButtonUpFcn'回调属性的用法。这两个函数用于将文本放入坐标轴对象中，有点类似于 Matlab 函数 gtext。除此以外，这两个函数还添加了一项新功能：*用户可以在文本放入后用鼠标拖动到理想的放置位置*。mmtext6 函数用于演示 Matlab 6 及更早的版本中回调字

符串的用法，而 mmtext7 演示了 Matlab 7 中函数句柄回调的用法。下面是 mmtext6 函数的代码：

```
function h=mmtext6(arg)
%MMTEXT6 Place and Drag Text with Mouse.
% MMTEXT6 waits for a mouse click on a text object in the current figure
% then allows it to be dragged while the mouse button remains down.
%
% MMTEXT6('whatever') places the string 'whatever' on the current axes
% and allows it to be dragged with the mouse
%
% Ht=MMTEXT6('whatever') returns the handle to the text object.
%
% MMTEXT6 becomes inactive after the move is complete or no text
% object is selected.

if nargin==0, arg=0; end
if ischar(arg)  % user entered text to be placed
    Ht=text('Units','normalized',...
            'Position',[.5 .5],...
            'String',arg,...
            'HorizontalAlignment','center',...
            'VerticalAlignment','middle');
    if nargout>0, h=Ht; end
    mmtext6(0)      % call mmtext6 again to drag it
 elseif arg==0  % initial call, select text for dragging
     Hf=get(0,'CurrentFigure');
     if isempty(Hf)
        error('No Figure Window Exists.')
     end
     set(Hf,'BackingStore','off',...   % Speed up rendering
            'DoubleBuffer','on',...    % get rid of screen flicker
            'WindowButtonDownFcn','mmtext6(1)')
     figure(Hf)      % bring figure forward
 elseif arg==1 & strcmp(get(gco,'type'),'text')  % text object selected
     set(gco,'Units','data',...
             'HorizontalAlignment','left',...
             'VerticalAlignment','baseline');
     set(gcf,'Pointer','topr',...
             'WindowButtonMotionFcn','mmtext6(2)',...
             'WindowButtonUpFcn','mmtext6(99)')
elseif arg==2   % dragging text object
    cp=get(gca,'CurrentPoint'); % get current mouse point
    set(gco,'Position',cp(1,1:3)) % move text to the current point
else    % mouse button up or incorrect object selected, reset everything
   set(gcf,'WindowButtonDownFcn','',...
           'WindowButtonMotionFcn','',...
           'WindowButtonUpFcn','',...
           'Pointer','arrow',...
           'DoubleBuffer','off',...
           'BackingStore','on')
end
```

在上面的函数中，首先将图形的'WindowButtonDownFcn'回调属性设置为'mmtext6(1)'，这表明将 mmtext6 函数作为回调函数。然后，使用 gco 设置选中的文本句柄。之后，将图形的'WindowButtonMotionFcn'回调属性设置为'mmtext6(2)'，'WindowButtonUpFcn'回调属性设置为'mmtext6(99)'。mmtext 函数的 3 个不同的数值参数表明需要其执行不同的操作，这些操作都已在函数体内定义了。最后，当移动文本结束后，所有的回调字符串都被设置为空字符串，表明一经移动好的文本就不能再进行选择和移动了。

mmtext7 与 mmtext6 功能类似，只不过它只能用于 Matlab 7 中，其代码如下：

```
function h=mmtext7(arg)
%MMTEXT7 Place and Drag Text with Mouse.
% MMTEXT7 waits for a mouse click on a text object in the current figure
% then allows it to be dragged while the mouse button remains down.
%
% MMTEXT7('whatever') places the string 'whatever' on the current axes
% and allows it to be dragged with the mouse
%
% Ht=MMTEXT7('whatever') returns the handle to the text object.
%
% MMTEXT7 becomes inactive after the move is complete or no text
% object is selected.

if nargin==0  % call subfunction to set up string drag
    local_mmtext_init
elseif ischar(arg)  % user entered text to be placed
    Ht=text('Units','normalized',...
            'Position',[.5 .5],...
            'String',arg,...
            'HorizontalAlignment','center',...
            'VerticalAlignment','middle');
    if nargout>0, h=Ht; end
    local_mmtext_init      % call subfunction to set up string drag
else
    error('String Input Expected.')
end

%-------------------------------------------------------------------
function local_mmtext_init(cbo,eventdata)
% two input arguments required even if not used
Hf = get(0,'CurrentFigure');
If isempty(Hf)
    error('No Figure Window Exists.')
end
set(Hf,'BackingStore','off',...   % speed up rendering
       'DoubleBuffer','on',...    % get rid of screen flicker
       'WindowButtonDownFcn',@local_mmtext_down)
figure(Hf)   % bring figure forward

%-------------------------------------------------------------------
function local_mmtext_down(cbo,eventdata)
```

```
% two input arguments required even if not used

if strcmp(get(gco,'type'),'text')  % text object selected

    set(gco,'Units','data',...
            'HorizontalAlignment','left',...
            'VerticalAlignment','baseline');
    set(gcf,'Pointer','topr',...
            'WindowButtonMotionFcn',@local_mmtext_drag,...
            'WindowButtonUpFcn',@local_mmtext_up)
else
    local_mmtext_up   % reset everything
end

%------------------------------------------------------------------
function local_mmtext_drag(cbo,eventdata) % dragging text object
% two input arguments required even if not used

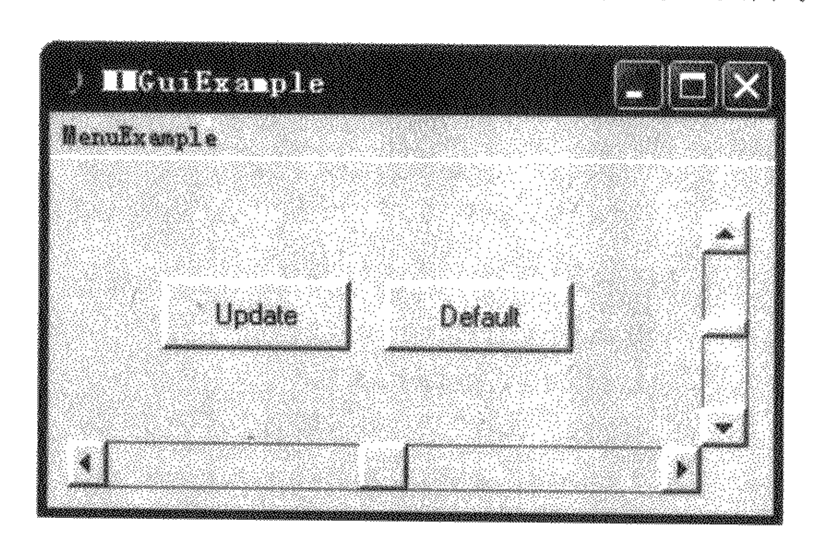
cp = get(gca,'CurrentPoint');  % get current mouse point
set(gco,'Position',cp(1,1:3))  % move text to the current point

%------------------------------------------------------------------
function local_mmtext_up(cbo,eventdata)  % reset everything
% two input arguments required even if not used

set(gcf,'WindowButtonDownFcn','',...
        'WindowButtonMotionFcn','',...
        'WindowButtonUpFcn','',...
        'Pointer','arrow' ,...
        'DoubleBuffer','off',...
        'BackingStore','on')
```

上面的函数句柄回调实例 mmtext7 与字符串回调实例 mmtext6 并没有本质区别，但阅读起来更加容易。两个函数中设置和清除回调的方式和时机完全相同。惟一的不同是在 mmtext7 中，切换编程结构由子函数回调取代了。

下面我们再介绍一个 GUI 的实例，该实例的 GUI 界面如下所示：



由上图可以看到，该 GUI 界面中包含一个 uimenu 对象、两个按钮类型的 uicontrol 对象、两个滑动杆类型的 uicontrol 对象。用户上下拖动纵向的滑动杆就使窗口上下移动，左右拖动横向的滑动杆就使窗口左右移动。因此，滑动杆（或称为滚动轴）可以使图形在整

个当前有效窗口内移动。另外，图中的 Default 按钮将图形重新恢复成默认的大小和居中的显示位置。如果用户手动拖动鼠标，按下 Update 按钮就可以及时更新滑动杆的指示符用以反映当前的图形位置。图中的 MenuExample 菜单仅包含一个 Close 菜单项，用于关闭 GUI 图形。下面给出了创建该 GUI 图形的具体代码:

```
function mmguiexample
%MMGUIEXAMPLE Example GUI in Mastering MATLAB 7

% Build GUI

set(0,'Units','pixels');
Ssize = get(0,'ScreenSize'); % get screen size so gui can be centered

H.gui = dialog('WindowStyle','normal',...  % Figure object with good
               'Resize','on',...            % properties for a gui
               'Name','MMGuiExample',...    % Add modifications
               'Units','pixels',...
               'Position',[(Ssize(3)-310)/2 (Ssize(4)-150)/2 310 150]);

DefOutPos = get(H.gui,'OuterPosition'); % undocumented figure property
set(H.gui,'UserData',DefOutPos) % store default outer position here

Hm = uimenu('Parent',H.gui,'Label','MenuExample'); % create top level menu
    uimenu('Parent',Hm,... % add 'Close' menu item to top level menu
           'Label','Close',...
           'Callback','close(gcbf)');        % simple string callback

H.Hslider = uicontrol('Style','slider',...   % horizontal slider
                      'Parent',H.gui,...
                      'Units','pixels',...
                      'Position',[10 10 270 20],...
                      'Min',20,'Max',Ssize(3)-DefOutPos(3)-20,...
                      'Value',DefOutPos(1),...
                      'Callback',{@local_Hslider,H});

H.Vslider = uicontrol('Style','slider',... % vertical slider
                      'Parent',H.gui,...
                      'Units','pixels',...
                      'Position',[280 30 20 100],...
                      'Min',20,'Max',Ssize(4)-DefOutPos(4)-20,...
                      'Value',DefOutPos(2),...
                      'Callback',{@local_Vslider,H});

H.Update = uicontrol('Style','pushbutton',... % Update pushbutton
                     'Parent',H.gui,...
                     'Units','pixels',...
                     'Position',[50 70 80 30],...
                     'String','Update',...
                     'Callback',{@local_Update,H});

H.Default = uicontrol('Style','pushbutton',... % Default pushbutton
                      'Parent',H.gui,...
                      'Units','pixels',...
```

```
                        'Position',[145 70 80 30],...
                        'String','Default',...
                        'Callback',{@local_Default,H});
%-----------------------------------------------------------
% Subfunction callbacks
%-----------------------------------------------------------
function local_Hslider(cbo,eventdata,h)
% Callback for horizontal slider
% Move gui figure horizontally
% Slider value contains desired outer left position

SliderValue = get(cbo,'Value');
pos = get(h.gui,'OuterPosition');
set(h.gui,'OuterPosition',[SliderValue pos(2:4)])

%-----------------------------------------------------------
function local_Vslider(cbo,eventdata,h)
% Callback for vertical slider
% Move gui figure vertically
% Slider value contains desired outer bottom position

SliderValue = get(cbo,'Value');
pos = get(h.gui,'OuterPosition');
set(h.gui,'OuterPosition',[pos(1) SliderValue pos(3:4)])

%-----------------------------------------------------------
function local_Update(cbo,eventdata,h)
% Callback for Update pushbutton
% Update slider values to reflect current GUI position
% This button is only needed if the user drags the GUI
%  window manually with the mouse

OutPos = get(h.gui,'OuterPosition');
set(h.Hslider,'Value',OutPos(1))
set(h.Vslider,'Value',OutPos(2))

%-----------------------------------------------------------
function local_Default(cbo,eventdata,h)
% Callback for Default pushbutton
% Return GUI position vector to default value

defoutpos = get(h.gui,'UserData'); % retrieve default outer position
set(h.gui,'OuterPosition',defoutpos)
set(h.Hslider,'Value',defoutpos(1))
set(h.Vslider,'Value',defoutpos(2))
```

在上面的函数中，除了用于关闭图形的菜单项外，其他的回调都是通过函数句柄指定的。函数中的每一个 GUI 组件都通过注释进行了详细说明。由于函数句柄具有强大的功能，因此，上述函数没有考虑使用字符串回调。另外需要注意的是，上述代码仅仅是创建 GUI 的众多方法中的一种。在上例中，水平和垂直滑动杆的句柄仅仅被保存下来，而没有使用，是因为在其回调函数中的 cbo 变量已经包含了这两个句柄。函数中的'Userdata'属性用于保存默认的图形位置向量，该变量也可以作为一个附加参数传递给子函数 local_Default。

32.12 图形用户接口设计环境（GUIDE）

GUIDE 是 Matlab 提供的一个 GUI 工具，用于快速、便捷、可靠地创建用户自己的 GUI。函数 guide 针对用户创建、定位、对齐和重置用户接口对象提供了强大的支持：首先，该函数提供了属性编辑器和查看器，用于列出对象的属性，使得用户可以交互地修改这些属性的值；另外，该函数还提供了一个菜单编辑器，用于交互地编辑和重新布置用户定义的下拉菜单和上下文菜单。除此以外，GUIDE 还提供了一个开发 GUI 的交互式方法，该方法可以显示出 GUI 的几何布局，能够大大降低 GUI 开发和执行时的难度，另外还能使 GUI 的结构在不同 GUI 之间保持稳定。GUIDE 将图形用户接口保存在一个 FIG 文件中，并创建一个 M 文件初始化 GUI 并保存用户回调的代码。GUIDE 对 Matlab 版本的依赖性很强，它随着 Matlab 版本的不断升级而变化。在 Matlab 7 中，GUIDE 可以创建更加容易调试和稳定性更强的 GUI。不过，Matlab 7 的 GUIDE 也有一个不太完美的地方，那就是用户需要了解所有 GUI 对象之间的细微差别，才能创建出一个性能优越、功能符合的 GUI。例如，用户必须对'HitTest'、'Interruptible'、'BusyAction'等句柄图形属性有充分了解，同时还要对诸如单击按钮排序、事件序列、回调等概念有深刻理解。

32.13 小结

并不是每个用户都需要进行图形用户界面设计。但是，如果用户确实需要一个 GUI，那么就可以在 Matlab 中创建这个 GUI。利用 Matlab 提供的强大数值和图形功能，GUI 可以成为展示和浏览大量的科学数据的强大工具。

下表总结了 Matlab 中提供的 GUI 函数，其中既有本章讲到的函数，也包括一些在本章中没有涉及到的函数。

函数	描述
uibuttongroup	用于管理'radiobutton'和'togglebutton'类型控件对象的用户接口容器对象
uicontainer	创建用户接口容器对象
uicontrol	创建用户接口控件对象
uimenu	创建用户接口菜单对象
uicontextmenu	创建用户接口上下文菜单对象
uipanel	创建用户接口面板对象
uitoolbar	创建用户接口工具栏对象
uipushtool	创建瞬间接触按压按钮控件对象
uitoggletool	创建开关按钮控件对象
uitable	创建用户接口表格对象
uitree	创建用户接口树对象
uitreenode	创建用户接口树节点对象
drawnow	处理所有未完成的图形事件，并立即更新屏幕

（续表）

函数	描述
gcbf	获取回调图形句柄
gcbo	获取回调对象句柄
dragrect	用鼠标拖动时所显示的矩形
rbbox	捕获橡皮圈框的位置
selectmoveresize	交互式地对坐标轴和控件进行选择、移动和重置大小操作
waitforbuttonpress	等待在一个图形上按下键盘上的按键或按下鼠标键
waitfor	停止程序执行，等待一个事件发生
uiwait	停止程序执行，等待恢复信号
uiresume	恢复一个被停止的 M 文件执行
uistack	控制对象的层叠顺序
uisuspend	暂停一个图形的交互状态
uirestore	恢复一个图形的交互状态
uiclearmode	清除当前的交互模式
guide	图形用户接口设计环境
inspect	查看对象属性
ishandle	判断是否是一个有效的对象句柄，若是，返回 True，否则，返回 False
isprop	判断是否是一个有效的对象属性，若是，返回 True，否则，返回 False
align	对齐控件和坐标轴
propedit	打开属性编辑器 GUI
makemenu	创建菜单对象结构
umtoggle	翻转一个菜单项的选中状态
getpixelposition	获取以像素表示的对象位置
setpixelposition	设置以像素表示的对象位置
getptr	获取图形指针
setptr	设置图形指针
hidegui	隐藏或显示 GUI
movegui	将 GUI 窗口移动到屏幕上的一个指定位置
guidata	保存或获取应用数据
getappdata	获取与一个 GUI 相关的应用数据
setappdata	设置与一个 GUI 相关的应用数据
rmappdata	删除与一个 GUI 相关的应用数据
isappdata	判断一个已命名的应用数据是否存在，如果存在，返回 True，否则，返回 False
guihandles	创建句柄结构体
overobj	获取鼠标指针所在的对象的句柄
popupstr	获得弹出菜单选择字符串
remapfig	变换图形中对象的位置

关于 GUI 创建的更详细的信息请读者参考相应的 Matlab 在线帮助文档。

Chapter 33

Matlab 类和面向对象编程

Matlab 提供了许多基本的数据类型，这些数据类型又被称为类。例如，我们通常使用的数字数组都是双精度数组，那么由这些数组就可以构成一个称为 double 的类。同样，字符串数据类型也是一个类，称为 char 类。我们可以使用 class 函数查看一个数字或数组属于什么类，如下例所示：

```
>> pi                    % a simple double
ans =
      3.1416

>> class(pi)
ans =
double

>> s ='pi'               % a simple string
s =
pi

>> class(s)
ans =
char
```

Matlab 的基本数据类型或类包括 double、char、logical、cell 和 struct。这些数据类型都是 Matlab 中最常用的数据类型。另外，Matlab 还包含了一些我们不常用到的类，如 function、handle、inline、java、single 以及一系列整数数据类型。

Matlab 针对每一个类都定义了相应的操作和运算。例如，Matlab 为 double 类定义了加法运算，但该运算不能用于 char 类和 cell 类，如下例所示：

```
>> x = pi+2
x =
          5.1416

>> y = 'hello' + 'there'
y =
   220     205     209     222     212

>> {'hello' 'there'}+{'sunny' 'day'}
??? Function 'plus' is not defined for values of class 'cell'.
```

在上面的例子中，将两个字符串相加（y = 'hello' + 'there'），结果将生成一个数字数组，而不是一个字符串。由于这两个字符串长度相等，因此 Matlab 没有对该运算报错，而是将'hello'和'there'都转换成相应的 ASCII 数字，然后再依次将每个数字相加。从表面上来看，Matlab 似乎能对字符串进行加法运算，但实际上，Matlab 是在将字符串转换成 double 类之后才进行加法运算的。Matlab 在进行字符串相加时进行隐含的数据类型（类）转换，是为了计算方便，而不代表 Matlab 对字符串定义了加法运算。在上例的最后一条语句中，用户试图强行将两个单元数组相加，结果产生了一个错误。

从 Matlab 5 开始，Matlab 中增加了两项新功能：①可以为基本数据类型或类定义新的运算。②可以创建用户自定义的数据类型或类。这两种创建和使用数据类型的功能被称为面向对象编程（OOP），其中的每个数据类型或类中的变量被称为对象。基于对象的运算通常通过数据封装和操作符与函数过载的方法完成。在 Matlab 中，同样使用了面向对象编程工具使用的一些术语，包扩运算符和函数过载、数据封装、方法、继承、集合等。本章将针对这些术语以及面向对象编程的基本规则进行讨论。

33.1 重载

在我们深入了解面向对象编程的详细内容并创建新的变量类之前，首先看一看如何在 Matlab 中对标准类进行重载。对标准类的重载与对用户生成类的重载的方法是一样的。因此，理解了标准类的重载，用户就可以理解用户生成类的重载。

当 Matlab 的程序解释器遇到一个运算符（例如加法运算符'+'）或一个具有一个或多个输入参数的函数时，Matlab 会判断运算符或函数的参数的数据类型（也就是类），并按照其内部定义的规则进行运算。例如，如果是加法运算符，就意味着：如果参数是数字或可以被转换成数字的值（例如字符串），就计算参数的数字之和。如果 Matlab 或用户对运算符或函数的内部规则进行了重新定义，那么就称该运算符或函数被“重载”了。

> 运算符和函数过载允许用户重新定义 Matlab 在遇到此运算符或函数时所执行的运算。用于重新定义运算符和函数的规则或 M 文件的集合称为方法，这些文件本身通常被称为方法函数。

在 Matlab 中，对运算符和函数进行重新定义的规则通常都是一些 M 函数文件，这些文件一般不会直接存储在 Matlab 搜索路径中，但必须存储在 Matlab 搜索路径中某一目录的子目录中。因此，为了能够找到该子目录，Matlab 要求类的目录必须以“@class”来命名，其中 class 是“@class”中的 M 文件所应用的变量类的名称。Matlab 支持多个类子目录，也就是说，我们可以在 Matlab 路径下放置多个基于同一个类的“@class”子目录。当 Matlab 在类子目录中寻找函数时，会执行 Matlab 搜索路径所给出的命令，并且使用所找到的第一个符合要求的方法函数文件。

例如，如果目录“@char”是 Matlab 搜索路径的下级目录，此目录中的 M 文件就能够重新定义关于字符串的操作和函数运算。为了更好地说明，请读者阅读下面的 M 函数文件 plus.m：

```
function s=plus(s1,s2)
% Horizontal Concatenation for char Objects.

if ischar(s1)&ischar(s2)
    s=cat(2,s1(:).',s2(:).');
elseif isnumeric(s2)
    s=double(s1)+s2;
else
    error('Operator + Not Defined.')
end
```

如果上面的 M 函数文件存储在 Matlab 搜索路径的下级目录，那么字符串的加法将被重新定义为横向串联。例如，如果我们再次执行前面的语句：y = 'hello' + 'there' ，可以得到：

```
>> y = 'hello'+'there'
y =
hellothere
```

由于重载的原因，Matlab 不再将字符串转换成相应的 ASCII 代码进行数值加法运算，而是按顺序执行下面的操作：①对“+”两侧的字符串进行逐字分析。②在 Matlab 的搜索路径下寻找@char 子目录。③找到我们所创建的子目录，然后在其中寻找名为 plus.m 的 M 文件。④找到上面所给出的 plus.m 函数，将两个参数（'hello'和'there'）传递给与加法运算符相对应的重载函数，然后由重载函数决定所执行的运算。⑤最后将重载函数返回。

> 为了加快运算速度，Matlab 会在启动时将与类相联系的子目录存储在缓存中。因此，要想使上面的例子进行重载运算，必须要在创建子目录和重载 M 文件之后，重新启动 Matlab 或使用 rehash 命令，使 Matlab 能够将最新创建的类子目录和重载 M 文件存储在缓存中。

如果用户在两种不同的数据类型（例如 char 和 double）之间进行加法运算时，Matlab 会按照某种优先级顺序解释参数。由于所有的变量类型都具有相同的优先级，因此 Matlab 将把最左边的参数视为最高优先级，例如：

```
>> z = 2 + 'hello'
z =
    106    103    110    110    113
```

由于 double 和 char 类具有相同的优先级，因此，Matlab 将 double 视为最高优先级，认为加法是针对 double 类型的数字加法，并将“hello”转换成相应的 ASCII 值（double 类型），然后进行数字加法运算。但是，如果将上面的两个操作数的顺序互换，即：

```
>> z ='hello' + 2
```

z 的返回结果为：

```
z =
    106    103    110    110    113
```

此时，Matlab 将加法视为针对 char 类型的运算。由于本例的调用格式为 plus('hello',2)，plus.m 将根据 isnumeric(s2)判断出这是一个混合类运算，然后返回与 z = 2 + 'hello' 相同的结果。

如上所述，为了重载加法，我们在类子目录中定义了 plus.m 函数。为了能够支持其他运算符重载，Matlab 为许多运算符重载指定了函数名，如下表所示。

<table>
<tr><th>运算符</th><th>函数名称</th><th>说明</th></tr>
<tr><td>a+b</td><td>plus(a,b)</td><td>数字加法</td></tr>
<tr><td>a-b</td><td>minus(a,b)</td><td>数字减法</td></tr>
<tr><td>-a</td><td>uminus(a)</td><td>一元减号</td></tr>
<tr><td>+a</td><td>uplus(a)</td><td>一元加号</td></tr>
<tr><td>a .* b</td><td>times(a,b)</td><td>逐个元素相乘</td></tr>
<tr><td>a * b</td><td>mtimes(a,b)</td><td>矩阵乘法</td></tr>
<tr><td>a ./ b</td><td>rdivide(a,b)</td><td>逐个元素相右除</td></tr>
<tr><td>a .\ b</td><td>ldivide(a,b)</td><td>逐个元素相左除</td></tr>
<tr><td>a/b</td><td>mrdivide(a,b)</td><td>矩阵右除</td></tr>
<tr><td>a\b</td><td>mldivide(a,b)</td><td>矩阵左除</td></tr>
<tr><td>a .^ b</td><td>power(a,b)</td><td>逐个元素求幂</td></tr>
<tr><td>a ^ b</td><td>mpower(a,b)</td><td>矩阵求幂</td></tr>
<tr><td>a < b</td><td>lt(a,b)</td><td>小于</td></tr>
<tr><td>a > b</td><td>gt(a,b)</td><td>大于</td></tr>
<tr><td>a <= b</td><td>le(a,b)</td><td>小于或等于</td></tr>
<tr><td>a >= b</td><td>ge(a,b)</td><td>大于或等于</td></tr>
<tr><td>a ~= b</td><td>ne(a,b)</td><td>不等于</td></tr>
<tr><td>a = = b</td><td>eq(a,b)</td><td>等于</td></tr>
<tr><td>a & b</td><td>and(a,b)</td><td>逻辑与</td></tr>
<tr><td>a | b</td><td>or(a,b)</td><td>逻辑或</td></tr>
<tr><td>~a</td><td>not(a,b)</td><td>逻辑非</td></tr>
<tr><td>a:d:b</td><td>colon(a,d,b)</td><td rowspan="2">冒号运算符</td></tr>
<tr><td>a:b</td><td>colon(a,b)</td></tr>
<tr><td>a'</td><td>ctranspose(a)</td><td>共轭转置</td></tr>
<tr><td>a.'</td><td>transpose(a)</td><td>转置</td></tr>
<tr><td>[a b]</td><td>horzcat(a,b)</td><td>水平串联</td></tr>
<tr><td>[a; b]</td><td>vertcat(a,b)</td><td>垂直串联</td></tr>
<tr><td>a(s1,s2,…)</td><td>subsref(a,s)</td><td>下标引用</td></tr>
<tr><td>a(s1,s2,…)=b</td><td>subsasgn(a,s,b)</td><td>下标分配</td></tr>
<tr><td rowspan="2">b(a)</td><td>subsindex(a)</td><td>下标索引</td></tr>
<tr><td>display(a)</td><td>命令窗口输出</td></tr>
<tr><td>end</td><td>end(a,k,n)</td><td>end 的下标说明</td></tr>
</table>

我们仍以上面的 char 类型的计算为例，下面的函数将实现减法的重载。

```
functon s=minus(s1,s2)
% Subtraction for char Objects
% Delete occurrences of s2 in s1.

if ischar(s1)&ischar(s2)
    s=strrep(s1,s2,'');
elseif isnumeric(s2)
    s=double(s1)-s2;
else
    error('Operator - Not Defined. ')
end
```

按照上面的定义，减法将被解释成为从字符串中删除对应的字符串，如下例所示：

```
>> z ='hello' -'e'
z =
hllo

>> a = 'hello' - 2
a =
   102     99     106     106     109
```

同样，在混合类的情况下，Matlab 将按照与加法相同的规则运算。

当一个语句中出现多个运算符时，Matlab 会按照它通常的优先级顺序执行，即从表达式的左端向右端执行，如下例所示：

```
>> a ='hello' + ' ' + 'there'
a =
hello there

>> a -'e'
ans =
hllo thr

>> a = 'hello' + ' ' + ('there' -'e')
a =
hello thr
```

当一条语句将一个字符串赋给一个输出参数，并且语句不带分号时，Matlab 将在命令窗口中显示该字符串。在命令窗口中显示的字符串可以使用函数 display.m 进行重载。虽然 Matlab 默认的字符显示模式比较方便，但是也可以使用如下所示的函数对命令窗口显示进行重载。

```
function display(s)
% Display for char objects.

isloose=strcmp(get(0,'FormatSpacing'),'loose');
ssiz= size(s);
```

```
if isloose,disp(' '), end
disp(['A Character Array of Size: 'mat2str(ssiz)])
if isloose, disp(' '), end
```

上面给出的函数重新定义了命令窗口中显示字符串的模式，如下面的例子所示：

```
>> 'hello'
A character Array of Size: [1 5]

>> a = 'hello'+' '+'there'
A Character Array of Size:[1 11]

>> format loose
>> a
A Character Array of Size:[1 11]

>> format compact
>> s=char('hello','there')
A Character Array of Size:[2 5]
```

下面我们看一下 Matlab 是如何对运算符进行重载的。重载函数有一个共同的规则：重载函数必须与标准的 Matlab 函数同名，例如：

```
function s=cat(varargin)
% CAT Concatenate Strings as a Row.

if length(varargin)>1 & ~ischar(varargin{2})
   error('CAT Not Defined for Mixed Classes.')
else
   s =cat(2,varargin{:});
end
```

上述函数对字符串 cat 函数进行了重载。只有当 cat 函数的第一个参数是字符串时，才会调用重载函数。如果 cat 函数的第一个参数是数字，那么将会调用标准的 cat 函数，如下例所示：

```
>> cat('hello','there')          % call overloaded cat
ans =
hellothere
>> cat('hello',2)                % call overloaded cat
??? Error using ==> char/cat
CAT Not Defined for Mixed Classes.

>> cat(2,'hello')                % call built-in cat
ans =
hello
```

除了本节列出的运算符重载函数外，Matlab 还提供了几个实用的面向对象编程（OOP）函数，它们包括 methods、isa、class、loadobj、saveobj。其中，isa 和 class 函数用于帮助确定对象或变量的数据类型（即类），例如：

```
>> a = 'hello';
```

```
>> class(a)          % return class of argument
ans =
char
>> isa(a,'double')     % logical class test
ans =
    0
>> isa(a,'char')    % logical class test
ans =
    1
```

methods 函数用于返回与某一个类相关的方法或重载运算符和函数的列表，例如：

```
>> methods cell
Methods for class cell:

cell2struct    ismember       regexprep      str2func       strmatch
tril           ctranspose     lower          reshape        strcat
strncmp        triu           diag           permute        setdiff
strcmp         strncmpi       union          display        regexp
setxor         strcmpi        strtok         unique         intersect
regexpi        sort           strfind        transpose      upper
```

所显示的结果表明，对上述函数，当输入为单元数组时，Matlab 本身就定义了需要被调用的重载函数。重载函数对 Matlab 的基本函数进行了扩展，以适合单元数组参数。使用重载函数的好处是，无需对函数本身进行重写就可以使函数接受单元数组参数。

最后，无论用户何时在自己定义的类中调用 load 和 save 函数，如果存在 loadobj 和 saveobj 函数，那么 loadobj 和 saveobj 函数将代替 load 和 save 函数被调用。因此，用户在类的子目录中添加这些函数，可以利用它们在装载（Load）运算后或保存（Save）运算前修改用户定义的变量。

33.2　类的创建

运算符和函数重载是面向对象编程（OOP）的关键内容。在 Matlab 中，实现重载的方法很简单：将与某一变量类相关联的方法存储到属于 Matlab 搜索路径下级目录的一个类目录中即可，而方法本身是一个标准的 M 函数文件（当然，也可以是与 M 文件等价的标准的 P 文件或 MEX 文件）。用户定义的类将使用与标准类相同的途径创建和存储方法函数。本节将向读者演示如何创建用户定义的类。

要创建一个新的变量类，用户需要在 Matlab 搜索路径的下级目录创建一个@classname 类目录，并至少在其中提供两个 M 函数文件：第一个 M 函数文件是 classname.m，该文件用于定义在新类中变量的生成，因此文件称为构造器文件；第二个 M 函数文件是 display.m，该文件用于在命令窗口中显示新变量。如果除这两个文件外没有附加的方法文件，那么变量类将没有什么用处，但是该变量类却是一个合法存在的变量类。

在 OOP 的术语表中，构造器用于创建类的一个实例，该实例是一个使用类中方法的对象，这些方法说明了在实例对象存在时，类中的操作符和函数是如何重载的。

构造器函数 classname.m 是一个带有输入参数的标准函数调用，输入参数中包含了为

创建期望类型的输出变量所需的数据。为了获得最大的灵活性，构造器应能处理 3 种不同的输入参数集合。首先，由于 Matlab 中存在空字符串、空数组、空单元等，因此，构造器应该能够在无输入参数的情况下输出一个空变量。另外，如果向构造器传递了一个由同一构造器创建的相同类型的变量，构造器应该能够直接将它作为输出参数传递出去。最后，如果向构造器提供了用户创建的数据，那么构造器应该创建一个期望类型的变量。最后一种情况可以用来验证输入数据的变量类型是否正确。在构造器的函数定义中，用于创建期望类型的变量的数据存储在一个结构体的域中。一旦提供了该结构体的域，就可以通过调用 class 函数来创建新变量。例如，下面给出一个有理多项式对象类型的构造器函数：

```
function r=mmrp(varargin)
%MMRP Mastering MATLAB Rational Polynomial Object Constructor.
% MMRP(p) creates a polynomial object from the polynomial vector p
% with 'x' as the variable.
% MMRP(p,'s') creates the polynomial object using the letter 's' as
% the variable in the display of p.
% MMRP(n,d) creates a rational polynomial object from the numerator
% polynomial vector n and denominator polynomial d.
% MMRP(n,d,'s') creates the rational polynomial using the letter 's' as
% the variable in the display of p.
%
% All coefficients must be real.

[n,d,v,msg]=local_parse(varargin); % parse input arguments
if isempty(v) % input was mmrp so return it
    r=n;
else
  error(msg) % return error if it exists
  tol=100*eps;
  if length(d)==1 & abs(d)>tol % enforce scalar d=1
      r.n=n/d;
      r.d=1;
   elseif abs(d(1))>tol % make d monic if possible
      r.n=n/d(1);
      r.d=d/d(1);
   else                 % can't be made monic
      r.n=n;
      r.d=d;
   end
   r.v=v(1);
   r=class(r,'mmrp');   % create object from parts
   r=minreal(r);        % perform pole-zero cancellation
end
```

上例中，为了简化函数定义，对输入参数的分解是由一个名为 local_parse 的子函数处理的。该子函数没有在上面的代码中显示，但可以看到它返回 4 个输出：n、d、v、msg。其中，变量 n 和 d 是两个数字类型的行向量，分别包含有理多项式的分母和分子的系数；变量 v 包含用于显示多项式的字符串变量；变量 msg 包含了 local_parse 在遇到无效输入时返回的错误信息。

根据上面的构造函数，有理多项式的创建是由语句 r = class (r,'mmrp') 来完成的。class 函数的此种用法只在构造器本身中有效。在其他情况下，class 将会返回表示输入参数类型的字符串。上面给出的构造器可以处理 3 种情况下的输入参数：①如果不存在输入参数，则构造器将会返回 n、d 和 v 来创建一个空的有理多项式。②如果输入参数是一个有理多项式对象，那么会直接将此对象作为输出参数返回。③如果用户提供了参数数据，那么构造函数将会根据参数创建一个有理多项式对象。最简单的一类有理多项式是分母等于 1，以“x”为多项式变量的多项式。mmrp 中的最后一条语句将已创建的有理多项式传递给重载函数 minreal，该函数通过删除极点和零点返回对象的一个最小实现。

在构造器 M 文件中，让所有结构体的各个域都保持相同的顺序是非常重要的。因为，虽然两个结构体包含有相同的域，但如果域定义的顺序不同，那么这两个结构体也是不相等的。如果用户违反了这一规则，将会导致创建的对象运算时的不确定性。

给出了 mmrp 构造器后，我们可以用下面的代码定义与有理多项式相关的显示函数 display.m：

```
function display(r)
%DISPLAY Command Window Display of Rational Polynomial Objects.

loose=strcmp(get(0,'FormatSpacing'),'loose');
if loose, disp(' '), end
var=inputname(1);
if isempty(var)
   disp('ans =')
else
   disp([var ' ='])
end
nstr=mmp2str(r.n,r.v); % convert polynomial to string
nlen=length(nstr);
if length(r.d)>1 | r.d~=1
   dstr=mmp2str(r.d,r,v);
else
   dstr=[];
end
dlen=length(dstr);
dash='-';
if loose, disp(' '), end
if dlen % denominator exists
   m=max(nlen,dlen);
   disp('MMRP Rational Polynomial Object:')
   disp([blanks(ceil((m-nlen)/2)) nstr]);
   disp(dash(ones(1,m)));
   disp([blanks(fix((m-dlen)/2)) dstr]);
else
   disp('MMRP Rational Polynomial Object:')
   disp(nstr);
end
if loose, disp(' '), end
```

display.m 文件中，首先调用 mmp2str 函数将数字多项式向量和期望的输出变量转换成

字符串表达式格式。注意：mmp2str 函数不能存储在@mmrp 目录中，而要存储在 Matlab 搜索路径下的其他目录中。如果该函数存储在类目录下，Matlab 将会找不到该函数，这是因为，mmp2str 的参数分别是 double 和 char 类型，而不是 mmrp 类型。上一节讲到，只有在最左端或最高优先级的输入变量的类型与方法函数相匹配时，该方法函数才会被调用。

在一个方法函数中，也可以像构造器中的最后一条语句 r = minreal (r) 一样对对象进行运算，其中，右端的变量 r 是一个具有 mmrp 类型的对象。但有两个函数例外：当在方法函数中出现下标引用和下标赋值时，Matlab 不会调用相应的重载函数 subsref 和 subsasgn。这就允许用户可以更加自由地访问和处理方法函数中的类变量。

用户还可以通过直接寻址对象的结构体域（如 display.m 中所示），将其应用在对象所包含的数据上。在这种情况下，数据的类型将决定 Matlab 执行的操作。

> 注意：在方法函数之外，例如在命令窗口中，对象的域是无法被访问的，同样，字段的数量和名称也无法被确定，这种属性被称为数据封装（data encapsulation）。

下面的例子演示了有理多项式对象的创建和显示：

```
>> p = mmrp([1 2 3])
p =
MMRP Rational Polynomial Object:
x^2 + 2x^1 + 3

>> q = mmrp([1 2 3],[4 5 6],'z')
q =
MMRP Rational Polynomial Object:
0.25z^2 + 0.5z^1 + 0.75
------------------------
z^2 + 1.25z^1 + 1.5

>> r = mmrp(conv([1 2],[1 4]),conv([1 2],[1 3]))
r =
MMRP Rational Polynomial Object:
x^1 + 4
--------
x^1 + 3
```

如果不进行运算符和函数重载，有理多项式其实是没什么价值的。使用重载可以很方便地在 mmrp 对象上进行算术操作。例如，下面的 M 文件针对 mmrp 对象定义了加法、减法、乘法和除法运算（乘法和除法可以通过多种方法实现，这些方法将通过基于多项式的方法函数重载给出）。

```
function r=plus(a,b)
%PLUS Addition for Rational Polynomial Objects.

if isnumeric(a)
    rn=mmpadd(a*b.d,b.n);    % see chapter 20 for mmpadd
    rd=b.d;
    rv=b.v;
```

```
elseif isnumeric(b)
   rn=mmpadd(b*a.d,a.n);
   rd=a.d;
   rv=a.v;
else % both polynomial objects
   if ~isequal(a.d,b.d)
      rn=mmpadd(conv(a.n,b.d),conv(b.n,a.d));
      rd=conv(a.d,b.d);
   else
      rn=mmpadd(a.n,b.n);
      rd=b.d;
   end
   if ~strcmp(a.v,b.v)
      warning('Variables Not Identical')
   end
   rv=a.v;
end
r=mmrp(rn,rd,rv);    % create new MMRP object from results
```

```
function r=minus(a.b)
%MINUS Subtraction for Rational Polynomial Objects.

r=a+(-b); % use plus and uminus to implement minus
```

```
function r=uminus(a)
%UMINUS Unary Minus for Rational Polynomial Objects.

r=mmrp(-a.n,a.d,a.v);

function r=times(a,b)
%TIMES Dot Times for Rational Polynomial Objects.

a=mmrp(a); % convert inputs to mmrp if necessary
b=mmrp(b);
rn=conv(a.n,b.n);
rd=conv(a.d,b.d);
if ~strcmp(a.v,b.v)
   warning('Variables Not Identical')
end
rv=a.v;
r=mmrp(rn,rd,rv);       % create new MMRP object from results
```

```
function r=mtimes(a,b)
%MTIMES Times for Rational Polynomial Objects.

r=a.*b; % simply call times.m
```

```
function r=rdivide(a,b)
%RDIVIDE Right Dot Division for Rational Polynomial Objects.

a=mmrp(a); % convert inputs to mmrp if necessary
b=mmrp(b);
rn=conv(a.n,b.d);
rd=conv(a.d,b.n);
if ~strcmp(a.v,b.v)
   warning('Variables Not Identical')
end
rv=a.v;
r=mmrp(rn,rd,rv); % create new MMRP object from results
```

```
function r=mrdivide(a,b)
%MRDIVIDE Right Division for Rational Polynomial Objects.

r=a./b; % simply call rdivide.m
```

```
function r=ldivide(a,b)
%LDIVIDE Left Dot Division for Rational Polynomial Objects.

r=b./a; % simply call rdivide.m
```

```
function r=mldivide(a,b)
%MLDIVIDE Left Division for Rational Polynomial Objects.

r=b./a; % simply call rdivide.m
```

上面给出的几个方法函数毋须多加解释，它们可以用来执行简单的多项式运算。下面给出了使用这些方法函数的几个简单例子：

```
>> a = mmrp([1 2 3])
a =
MMRP Rational Polynomial Object:
x^2 + 2x^1 + 3

>> b = a + 2                 % addition
b=
MMRP Rational Polynomial Object:
x^2 + 2x^1 + 5

>> a - b                     % subtraction
ans =
MMRP Rational Polynomial Object:
-2

>> a + b                     % addition
ans =
MMRP Rational Polynomial Object:
```

```
2x^2 + 4x^1 + 8

>> 2*b                          % multiplication
ans=
MMRP Rational Polynomial Object:
2x^2 + 4x^1 + 10

>> a * b                        % multiplication
ans =
MMRP Rational Polynomial Object:
x^4 + 4x^3 + 12x^2 + 16x^1 + 15

>> b/2                          % division
ans =
MMRP Rational Polynomial Object:
0.5x^2 + x^1 + 2.5

>> 2/b                          % division
ans =
MMRP Rational Polynomial Object:
       2
--------------
x^2 + 2x^1 + 5

>> c = a/b                      % division
C =
MMRP Rational Polynomial Object:
x^2 + 2x^1 + 3
--------------
x^2 + 2x^1 + 5

>> d = c/(1+c)                  % mixed

d=
MMRP Rational Polynomial Object:
0.5x^2 + x^1 + 1.5
------------------
x^2 + 2x^1 + 4

>> (a/b)*(b/a)                  % mixed
ans=
MMRP Rational Polynomial Object:
1
```

Matlab 本身也提供了一些易于使用的多项式函数，其中许多函数都可以进行重载。例如，基本的 Matlab 函数 roots 和 zeros 就可以使用下面的 M 文件进行重载：

```
function [z,p]=roots(r)
%ROOTS Find Roots of Rational Polynomial Objects.
% ROOTS(R) returns the roots of the numerator of R.
% [Z,P]=ROOTS(R) returns the zeros and poles of R in
% Z and P respectively.

z=roots(r.n);
```

```
if nargout==2
    p=roots(r.d);
end
```

```
function z=zeros(r)
%ZEROS Zeros of a Rational Polynomial Object.

z=roots(r.n);
```

上面的方法函数 roots 在其内部调用了基本 Matlab 函数 roots，这是因为方法函数内部所需要的参数也是 double 类型的。而方法函数 zeros 则因其所带参数不同，将具有完全不同的两种含义。因此，OOP 的优势在于：一个函数可以有多种含义或使用范围，无需将它们都嵌入到同一个 M 文件中，而仅仅通过输入参数的类型决定调用该函数的哪个方法进行操作。

仿效句柄图形的用法，我们通常将多项式用在 set 和 get 函数中，作为一个独立的结构体类的域，如下例所示：

```
function set(r,varargin)
%SET Set Rational Polynomial Object Parameters.
% SET(R,Name,Value,...) sets MMRP object parameters of R
% described by the Name/Value pairs:
%
% Name           Value
% 'Numerator'    Numeric row vector of numerator coefficients
% 'Denominator   Numeric row vector of denominator coefficients
% 'Variable'     Character Variable used to display polynomial

if rem(nargin,2)~=1
    error('Parameter Name/Values Must Appear in Pairs.')
end
for i=2:2:nargin-1
    name=varargin{i-1};
    if ~ischar(name), error('Parameter Names Must be Strings.'), end
    name=lower(name(isletter(name)));
    value=varargin{i};
    switch name(1)
    case 'n'
        if ~isnumeric(value) | size(value,1)>1
            error('Numerator Must be a Numeric Row Vector.')
        end
        r.n=value;
    case 'd'
        if ~isnumeric(value) | size(value,1)>1
            error('Denominator Must be a Numeric Row Vector.')
        end
        r.d=value;
```

```
        case  'v'
            if ~ischar(value) | length(value)>1
                error('Variable Must be a Single Character.')
            end
            r.v=value;
        otherwise
            warning('Unknown Parameter Name')
        end
    end
    vname=inputname(1);
    if isempty(vname)
        vname='ans';
    end
    r=mmrp(r.n,r.d,r.v);
    assignin('caller',vname,r);
```

```
function  varargout = get (r,varargin)
% GET Get Rational Polynomial Object Parameters.
% GET(R,Name) gets the MMRP object parameter of R described by
% one of the following names:
%
% name           Description
% 'Numerator'    Numeric row vector of numerator coefficients
% 'Denominator'  Numeric row vector of denominator coefficients
% 'Variable'     Character Variable used to display polynomial
%
% [A,B,...]=get(R,NameA,NameB,...)    returns multiple parameters
% in the corresponding output arguments.

if (nargout+(nargout==0))~=nargin-1
    error('No. of Outputs Must Equal No. of Names.')
end
for i=1:nargin-1
    name=varargin{i};
    if ~ischar(name), error('Parameter Names Must be Strings.'), end
    name=lower(name(isletter(name)));
      switch name(1)
      case  'n'
         varargout{i}=r.n;
      case  'd'
         varargout{i}=r.d;
      case  'v'
         varargout{i}=r.v;
      otherwise
         warning('Unknown Parameter Name')
      end
end
```

上面的两个函数可以用于修改 mmrp 对象，或从 mmrp 对象中获取数据，如下例所示：

```
>> c % recall data
c =
MMRP Rational Polynomial Object:
x^2 + 2x^1 + 3
---------------
x^2 + 2x^1 + 5

>> n = get(c,'n')              % get numerator vector
n =
        1       2       3

>> set(c,'Numerator',[3 1])    % change numerator
>> c
c =
MMRP Rational Polynomial Object:
3x^1 + 1
-----------------
x^2 + 2x^1 + 5

>> class(c) % class and isa know about mmrp objects
ans =
mmrp
>> isa(c,'mmrp')
ans =
    1
```

33.3 下标

由于 Matlab 本质上是面向数组的编程工具，因此用户自定义的类也可以使用下标来索引数据。在 Matlab 7 中，用户自定义类可以使用 3 种下标索引：V(...)、V{...}和 V.field。这些索引结构可以出现在赋值语句的任意一侧。如果出现在赋值语句的右侧，表明将引用变量 V 中的元素；如果出现在赋值语句的左侧，表明将对变量 V 的某些部分进行赋值。上述两个使用下标的索引过程分别称为下标引用和下标赋值。当在对象上使用下标时，可以分别使用 subsref 和 subasgn 函数来控制下标引用和下标赋值。这两个函数也用到了重载操作，可能初看起来不好理解，因此，本节我们将对其进行专门介绍。另外，为了便于讨论，本节仍针对前一节创建的 mmrp 对象进行讲解。

有理多项式中的下标引用有两种很明显的方式：一种是对于一个有理多项式对象 R，利用 R(x)返回所有 x 数据点上 R 的值（其中 x 是一个数据数组）；另一种形式是利用 R('v')将显示多项式对象所用的变量更改为指定的字母（其中'v'是任何有效的单个字符）。

下面给出的 subsref 函数的帮助文档说明了如何使用下标编写一个重载的 subsref 方法：

```
B = SUBSREF(A,S) is called for the syntax A(I), A{I}, or A.I
when A is an object. S is a structure array with the fields:
     type -- string containing '()', '{}', or '.' specifying the
             subscript type.
```

```
     subs -- Cell array or string containing the actual subscripts.
For instance, the syntax A(1:2,:) invokes SUBSREF(A,S) where S is a
1-by-1 structure with S.type='()' and S.subs = {1:2,':'}. A colon
used as a subscript is passed as the string ':'.

Similarly,the syntax A{1:2} invokes SUBSREF(A,S) where S.type ='{}'
and the syntax A.field invokes SUBSREF(A,S) where S.type ='.' and
S.subs ='field'.

These simple calls are combined in a straightforward way for
more complicated subscripting expressions. In such cases
length(S) is the number of subscripting levels. For instance.
A(1,2).name(3:5) invokes SUBSREF(A,S) where S is 3-by-1 structure
array with the following values:
    S(1).type='()'       S(2).type='. '      S(3).type='()'
    S(1).subs={1,2}      S(2).subs='name'    S(3).subs={3:5}
```

根据上面的帮助文本，如果 R 是一个 mmrp 对象，那么 R(x)将会创建一个 S.type='( )'和 S.subs=x 的对象，其中 x 包含的是需要计算的 R 的位置，而不是简单的数组索引。同样，R('v')将会创建一个 S.type='( )'和 S.subs='v'的对象。除了上述两种方式外，其他创建方式都将会产生错误。

因此，根据上面的描述，我们可以用下面的代码给出 subsref 函数的重载方法函数：

```
function y=subsref(r,s)
%SUBSREF(R,S) Subscripted Reference for Rational Polynomial Objects.
% R('z') returns a new rational polynomial object having the same
% numerator and denominator, but using the variable 'z'.
%
% R(x) where x is a numerical array, evaluates the rational polynomial
% R at the points in x, returning an array the same size as x.

if length(s)>1
      error('MMRP Objects Support Single Arguments Only.')
end
if strcmp(s.type,'()') % R(x) or R('v')
   arg=s.subs{1};
   argc=class(arg);
   if strcmp(argc,'char')
      if strcmp(arg(1),':')
         error('MMRP Objects Do Not Support R(:).')
      else
         y=mmrp(r.n,r.d,arg(1)); % change variables
      end
   elseif strcmp(argc,'double')
      if length(r.d)>1
         y=polyval(r.n,arg)./polyval r.d,arg);
      else
         y=polyval(r.n,arg);
      end
```

```
    else
       error('Unknown Subscripts.')
    end
else % R{ } or R.field
    error('Cell and Structure Addressing Not Supported.')
end
```

下面给出了使用上述重载方法的一些例子：

```
>> c % recall data
c =
MMRP Rational Polynomial Object:
3x^1 + 1
--------------
x^2 + 2x^1 +5

>> c = c('t') % change variable
c =
MMRP Rational Polynomial Object:
3t^1 + 1
--------------
t^2 + 2t^1 + 5

>> x = -2:2
x =
    -2      -1       0       1       2
>> c(x)      % evaluate c(x)
ans =
        -1      -0.5         0.2      0.5      0.53846

>> c{3}      % try cell addressing
??? Error using ==> mmrp/subsref
Cell and Structure Addressing Not Supported.

>> c.n       % Try field addressing
??? Error using ==> mmrp/subsref
Cell and Structure Addressing Not Supported.
```

前面讲到，在方法函数体外，对象的域结构是隐藏不可见的。因此最后一条语句 c.n 是不能访问 n 域的，无法返回多项式对象 c 的分子行向量。要想添加这一功能，必须对上面的 subsref 方法进行修改，明确声明可以对 n 域进行访问，修改后的 subsref 方法文件如下：

```
function y=subsref(r,s)
%SUBSREF(R,S) Subscripted Reference for Rational Polynomial Objects.
% R('z') returns a new rational polynomial object having the same
% numerator and denominator, but using the variable 'z'.
%
% R(x) where x is a numerical array, evaluates the rational polynomial
% R at the points in x, returning an array the same size as x.
```

```
%
% R.n returns the numerator row vector of R.
% R.d returns the denominator row vector of R.
% R.v returns the variable associated with R.

if length(s)>1
     error('MMRP Objects Support Single Arguments Only.')
end
if strcmp(s.type,'()') % R( )
   arg=s.subs{1};
   argc=class(arg);
   if strcmp(argc,'char')
        if strcmp(arg(1),':')
            error('MMRP Objects Do Not Support R(:).')
        else
            y=mmrp (r.n,r.d,arg(1));
        end
    elseif strcmp(argc,'double')
        if length(r.d)>1
            y=polyval(r.n,arg)./polyval(r.d,arg);
        else
            y=polyval(r.n,arg);
        end
    else
        error('Unknown Subscripts.')
    end
elseif strcmp(s.type,'.') % R.field
    arg=lower(s.subs);
    switch arg(1)
    case 'n'
       y=r.n;
    case 'd'
       y=r.d;
    case 'v'
       y=r.v;
    otherwise
        error('Unknown Data Requested.')
    end
else % R{ }
    error('Cell Addressing Not Supported.')
end
```

此时，我们就可以使用 subsref 方法访问 c 中的域了，如下所示：

```
>> c.n % return numerator
ans =
    3       1

>> c.v % return variable
ans =
x
```

```
>> c.nadfdf % only first letter is checked
ans =
    3       1

>> c.t      % not n,d, or v
??? Error using ==>mmrp/subsref
Unknown Data Requested.

>> c.d(1:2) % we didn't include subaddressing in subsref
??? Error using ==> mmrp/subsref
MMRP Objects Support Single Arguments Only.
```

如前所述，当下标引用和下标赋值出现在方法函数体内时，并不会隐含调用重载函数 subsref 和 subsasgn。这一点可以通过重载 Matlab 多项式求值函数 polyval 来描述，如下所示：

```
function y = polyval(r,x)
%POLYVAL Evaluate Rational Polynomial Object.
% POLYVAL(R,X) evaluates the rational polynomial R at the
% values in X.

if isnumeric(X)
    %y=r(x);          % what we'd like to do, but can't
    S.type='()';
    S.subs={x};
    y=subsref(r,S); % must call subsref explicitly
else
    error('Second Input Argument Must be Numeric.')
end
```

由于 subsref 函数是专门针对 mmrp 对象编写的，因此多项式求值只需执行 R(x)即可，其中 R 是 mmrp 对象，x 包含了需要求值的 R 的位置。有时候我们希望在 polyval 函数体内部也能执行这样的操作，即在 polyval 中只使用一条语句 y=r(x)，就可以使 Matlab 隐含地调用 subsref 方法来计算有理多项式的值。但实际上 Matlab 是无法在 polyval 方法中隐含调用 subsref 或 subsasgn 函数的。不过，如果用户要强制进行该操作，可以像访问域那样在方法函数中明确声明要调用 subsref 或 subsasgn 函数。

有理多项式中的下标赋值只有一个很明显的方式：对于一个有理多项式对象 R，利用 R(1,p)=v 对其分子多项式进行赋值，其中 p 是一个数字向量，用于确定变量的幂次，v 是一个和 p 等长的数字向量，它包含了与 p 相对应的分子多项式变量的系数。同样，R(2,q)=w 用于对分母多项式进行赋值，其用法及意义与分子多项式相同。

下面给出的 subsasgn 函数的帮助文档说明了如何使用下标编写一个重载的 subsasgn 方法：

```
A = SUBSASGN(A,S,B) is called for the syntax A(I)=B, A{I}=B, or
A.I=B when A is an object. S is a structure array with the fields:
     type -- string containing '()', '{}', or '.' specifying the
```

```
            subscript type.
     subs --Cell array or string containing the actual subscripts.
For instance, the syntax A(1:2,:)=B calls A=SUBSASGN(A,S,B) where
S is a 1-by-1 structure with S.type='()' and S.subs = {1:2,':'}.  A
colon used as a subscript is passed as the string ':'.

Similarly, the syntax A{1:2}=B invokes A=SUBSASGN(A,S,B) where
S.type='{}' and the syntax A.field=B invokes SUBSASGN(A,S,B) where
S.type='.' and S.subs='field'.

These simple calls are combined in a straightforward way for
more complicated subscripting expressions. In such cases
length(S) is the number of subscripting levels. For instance,
A(1,2).name(3:5)=B invokes A=SUBSASGN(A,S,B) where S is 3-by-1
structure array with the following values:
      S(1).type='()'        S(2).type='.'        S(3).type='()'
      S(1).subs={1,2}       S(2).subs='name'     S(3).subs={3:5}
```

根据上面的帮助文档和所要求的下标赋值，我们可以创建如下的 subsasgn 重载方法函数：

```
function a=subsasgn(a,s,b)
%SUBSASGN Subscripted assignment for Rational Polynomial Objects.
%
% R(1,p)=C sets the coefficients of the Numerator of R identified
% by the powers in p to the values in the vector C.
%
% R(2,p)=C sets the coefficients of the Denominator of R identified
% by the powers in p to the values in the vector C.
%
% R(1,:) or R(2,:) simply replaces the corresponding polynomial
% data vector.
%
% For example, for the rational polynomial object
%           2x^2 + 3x + 4
% R(x)  = -------------------------
%         x^3 + 4x^2 + 5x + 6
%
% R(1,2)=5           changes the coefficient 2x^2 to 5x^2
% R(2,[3 2])=[7 8]  changes x^3 + 4x^2 to 7x^3 + 8x^2
% R(1,:)=[1 2 3]    changes the numerator to x^2 + 2x + 3

if length(s)>1
       error('MMRP Objects Support Single Arguments Only.')
 end
if strcmp(s.type,'()') % R(1,p) or R(2,p)
   if length(s.subs)~=2
      error('Two Subscripts Required.')
   end
   nd=s.subs{1}; % numerator or denominator
   p=s.subs{2};  % powers to modify
```

```
      if ndims(nd)~=2 | length(nd)~=1 | (nd~=1 & nd~=2)
         error('First Subscript Must be 1 or 2.')
      end
      if isnumeric(p) & ...
         (ndims(p)~=2 | any(p<0) | any(fix(p)~=p))
         error('Second Subscript Must Contain Nonnegative Integers.')
      end
      if ndims(b)~=2 | length(b)~=prod(size(b))
         error('Right Hand Side Must be a Vector.')
      end
      b=b(:).';          % make sure b is a row
      p=p(:)';      % make sure p is a row
      if ischar(p)     & length(p)==1 & strcmp(p,':') % R(1,:) or R(2,:)
         if nd==1      % replace numerator
            r.n=b;
            r.d=a.d;
         else      % replace denominator
            r.n=a.n;
            r.d=b;
         end
      elseif isnumeric(p) % R(1,p) or R(2,p)
         plen=length(p);
         blen=length(b);
         nlen=length(a.n);
         dlen=length(a.d);
         if plen~=blen
             error('Sizes Do Not Match.')
         end
         if nd==1  % modify numerator
            r.d=a.d;
            rlen=max(max(p)+1,nlen);
            r.n=zeros(1,rlen);
            r.n=mmpadd(r.n,a.n);
            r.n(rlen-p)=b;
         else      % modify denominator
            r.n=a.n;
            rlen=max(max(p)+1,dlen);
            r.d=zeros(1,rlen);
            r.d=mmpadd(r.d,a.d);
            r.d(rlen-p)=b;
         end
      else
         error('Unknown Subscripts.')
      end
   else % R { } or R.field
      error('Cell and Stucture Addressing Not Supported.')
   end
   a=mmrp(r.n,r.d,a.v);
```

下面给出了使用上述重载方法的一些例子：

```
>> a = mmrp([3 1],[1 2 5 10]) % create test object
a =
MMRP Rational Polynomial Object:
      3x^1 + 1
----------------------
x^3 + 2x^2 + 5x^1 + 10

>> a(1,:) = [1 2 4]     % replace entire numerator
a =
MMRP Rational Polynomial Object:
   x^2 + 2x^1 + 4
----------------------
x^3 + 2x^2 + 5x^1 + 10

>> a(2,2) = 12          % replace x^2 coef in denominator
a =
MMRP Rational Polynomial Object:
   x^2 + 2x^1 + 4
-----------------------
x^3 + 12x^2 + 5x^1 + 10

>> a(1,0) = 0           % replace 4x^0 with 0x^0
a =
MMRP Rational Polynomial Object:
     x^2  +  2x^1
-----------------------
x^3 + 12x^2 + 5x^1 + 10

>> a(1,:)=a.d           % subsref and subsasn! (a.n and a.d cancel)
a =
MMRP Rational Polynomial Object:
1
```

33.4 转换器函数

前面的章节中讲到，double、char、logical 函数都可以用来将输入数据转换成与它们的函数名称相同的数据类型。例如：double('hello')可以将字符串'hello'转换成相应的 ASCII 代码。我们在使用这些函数时，为了使用方便，最好将这些转换器函数包含到一个类目录中。对于前面的 mmrp 对象而言，double 和 char 函数是使用最多的两个转换器函数。其中，double 方法用于提取多项式的分子和分母系数，char 方法则用于创建多项式的字符串表达式，即用 display.m 所显示的字符串。重载后的这两个转换器函数如下所示：

```
function [n,d]=double(r)
%DOUBLE Convert Rational Polynomial Object to Double.
% DOUBLE(R) returns a matrix with the numerator of R in
% the first row and the denominator in the second row.
% [N,D]=DOUBLE(R) extracts the numerator N and denominator D
% from the rational polynomial object R.
```

```
if nargout<=1 & length(r,d)>1
    nlen=length(r.n);
    dlen=length(r.d);
    n=zeros(1,max(nlen,dlen));
    n=[mmpadd(n,r.n);mmpadd(n,r.d)];
elseif nargout<=1
    n=r.n;
else % nargout==2
    n=r.n;
    d=r.d;
end
```

```
function [n,d,v]=char(r)
%CHAR Convert Rational Polynomial Object to Char.
% CHAR(R) returns a 3-row string array containing R in the
% format used by DISPLAY.M
% (N,D)=CHAR(R) extracts the numerator N and denominator D
% as character strings from the rational polynomial object R.
% [N,D,V]=CHAR(R) in addition returns the variable V.

if nargout<=1
  nstr=mmp2str(r.n,r.v);
nlen=length(nstr);
 if length(r.d)>1
     dash=,'';
     dstr=mmp2str(r.d,r.v);
     dlen=length(dstr);
     m=max(nlen,dlen);
     n=char([blanks(ceil((m-nlen)/2)) nstr]  ....
          dash(ones(1,m))  ....
          [blanks(fix((m-dlen)/2)) dstr]);
 else
     n=nstr;
 end
elseif nargout>1
   n=mmp2str(r.n); % converts polynomial to string
   d=mmp2str(r.d);
end
if nargout>2
     v=r.v;
end
```

33.5 优先级、继承和集成

Matlab 会自动给用户自定义的类赋予比 Matlab 内置类更高的优先级。因此，如果在操作符和函数体内部同时存在用户自定义的类和 Matlab 内置类，通常会调用用户自定义类中

的方法。对于只有一个用户自定义类的简单问题而言，该默认优先级就能完全满足计算需要。但如果存在多个用户定义的类，就需要有一种机制来控制不同用户定义类的优先级别。另外，用户有时候也可能希望强制使用用户自定义的类的优先级低于 Matlab 内置类。为此，Matlab 提供了 inferiorto 和 superiorto 用于控制和设置类的优先级顺序。需要说明的是，这两个函数只能在类的构造器函数中使用。这两个函数的参数都是一个字符串列表，该字符串列表指定了优先级高于或低于构造器所创建的类的各个类的名称。例如，superiorto('double')表明使所创建的对象类的优先级高于双精度变量；inferiorto('mmrp', 'char')则表明使所创建的对象类的优先级低于 mmrp 和 char 对象。

对于大型的编程项目而言，为诸多的对象类型创建一个分级层次关系，会给编程带来很多方便。在层次关系中，可以很方便地让一种对象类型从另一种对象类型中继承所需的方法。这样的话，用户只需要编写少数几个方法，主要的精力则在于对方法进行修改以实现继承。在面向对象编程中，继承其他类的属性的对象被称为子类，而被继承的类则称为父类。最简单的继承是子类从单个父类中继承方法的情况，这种继承被称为简单继承或单独继承。如果一个子类从多个父类中继承了方法，则被称为多重继承。

在简单继承中，通常会将父类的所有域都赋予子类，并为子类创建一个或多个它自己独有的域。因此，与父类相关的方法可以直接应用在子类对象上，而父类的方法却无法得到子类独有域的任何信息，更不可能使用这些域。同样，子类独有的方法也无法访问父类的域，而必须使用从父类继承来的方法访问父类的域。简单继承是最常见的继承，例如，控制工具箱（Control Toolbox）中的 lti 对象与其 5 个子类 tf、zpk、ss、dss、frd 之间就是简单继承关系。

在多重继承中，子类将拥有所有父类的所有域，另外还包括一个或多个它自己独有的域。与简单继承相同，父类和子类都不能直接访问对方的域。在一个子类具有多个父类的情况下，要判断在何种情况下继承哪个父类的属性和方法，不是三言两语可以说清的，有兴趣的读者可以参考相应的 Matlab 帮助文档。

在前面讲到 mmrp 类的例子中，对象的域中包含了 double 类和 char 类的数据。对象的域可以包含任意类型的数据，包括用户自定义的类型的数据。在面向对象编程中，这一现象通常被称为封装或集成。在进行封装时，操作符和函数的重载规则并不会发生改变。另外，在一个类的方法函数中，如果需要对其他类的域进行运算，也可以调用其他类的方法。

Chapter 34

Matlab 编程接口

Matlab 提供了多种与外部程序进行接口的方法。例如，在 Matlab 中可以通过使用 MEX 文件来调用 C 函数和 FORTRAN 子程序。此外，通过使用 Matlab 引擎（Engine），用户可以在 Matlab 中执行运算并将结果返回到 C 或 FORTRAN 程序中。Matlab 还提供了一些头文件和库文件用于创建和访问标准的 Matlab MAT 文件。使用 Matlab 内置的串行接口，用户可以直接将数据采集并载入到 Matlab 中。另外，Matlab 可以通过组件对象模型（COM）和动态数据交换（DDE）来使用 Java 的类、对象和方法，并与 PC 应用程序进行数据交换。Matlab 还可以作为一个 COM 自动化服务器与 Visual Basic（VB）应用程序或能够使用 Visual Basic for Application（VBA）的应用程序（例如 Microsoft Excel、PowerPoint、Word）进行通信。有关与 Java、DDE、ActiveX 等接口的内容将在后面的章节中介绍。本章将着重介绍 Matlab 与 C 和 FORTRAN 的接口。

Matlab 提供了一套非常丰富、功能强大的外部编程接口，本书所介绍的内容只是整个外部接口内容的"冰山一角"。目前，已有不少书籍专门介绍 Matlab 的应用程序接口（API）问题。本章和后面两章将通过一些简单的实例对 Matlab 的 API 特性进行指南性介绍。如果读者需要有关 Matlab 的更详尽的知识，可以参考相应的在线帮助文档。

如不特别声明，本章所讨论的内容和实例都是基于 UNIX 平台中的 C 编译器的，在需要的地方会讨论不同平台中的区别。本章中的实例代码都是使用 Linux i386 平台（2.6.6 版内核，glnx86 体系结构）上的 3.3.3 版 GCC 编译器进行开发和测试的。其中的 C 语言的例子是在 Windows 2000 Pentium PC 环境下（win32 体系结构）使用 Matlab 提供的 LCC 编译器来测试的，而 FORTRAN 代码则是在 Linux 平台下使用 3.3.3 版 G77 FORTRAN 编译器来测试的。

34.1 访问 Matlab 数组

要编写高效的 Matlab 接口程序，首先应该对 Matlab 的数组结构有一个基本了解，因为所有的 MEX 程序、MAT 程序或引擎程序都需要通过访问 Matlab 的数组来执行指定的操作。本节将对 Matlab 的数组存储方式进行简单介绍，并列出了一些在编程时可能用到的 Matlab 数组访问函数。

Matlab 数组

Matlab 只支持一种对象类型——Matlab 数组。所有的 Matlab 变量都以 Matlab 数组形式出现。Matlab 数组中的元素由下列基本数据类型之一：double，sparse，char，cell，object，或 8-，16-，32-，64 比特的有符号和无符号整数，或其他 Matlab 数组组成。所有的标量和复合元素（如字符串、向量、矩阵、单元数组、结构体）都被看作 Matlab 数组。

Matlab 按列存储数组中的数据。在 MEX 例 4 中的 mmcellstr 的 MEX 文件中，便利用这一特性将数据从一个字符串缓冲区中提取出来，并重新放到一个单独的字符串中。下面的示例代码反映了 Matlab 的这一数据组织形式：

```
>> x = [1 2 3; 4 5 6; 7 8 9]
x =
     1     2     3
     4     5     6
     7     8     9

>> size(x)
ans =
     3           3

>> x(:)'
ans =
     1     4     7     2     5     8     3     6     9
```

访问 Matlab 数组：mxArray

在 C 语言中，Matlab 数组被视为一种新的 C 数据类型——一个由 C 结构建立的被称为 mxArray 的对象。mmArray 结构保存了有关数组类型（如 double、sparse、cell 等）、数组维数和数据本身的信息。另外，mxArray 结构还保存了与特定类型相关的信息，如一个数字数组是实数类型还是复数类型，一个结构体或对象的域的数量和名称，稀疏矩阵的非零值索引和非零元素的最大数量等。通常，用来访问 mxArray 结构的特性和元素的函数名都以 mx 开头，称为 MX 函数。

数值矩阵用两个 double 类型（也可以是其他数据类型）的向量存储：一个向量存储实数部分，另一个向量存储虚数部分。因此，就需要使用两个指针来访问这些数据：一个指针指向实数向量（pr），另一个指针指向虚数向量（pi）。如果数组是实数数组，那么 pi 指针将为 NULL。字符串将用一个不含虚数部分的 16 位整数（即每字符 16 比特）存储。C 字符串通常以 NULL 结束，而 Matlab 字符串则不含 NULL，其长度一般从数组的维数中获得。

单元数组是多个数组的集合，通常需要用一个数据向量包含指向其他数组的指针。指针向量中的每个指针都被称为一个单元。结构体通常以折叠的单元数组存储，单元数组中的数据向量的每个元素都是一个域。每个域都和存储在 mxArray 的另一个结构体元素中的名字相关联。对象用具有特定名称（一般为类的名称）的结构来存储，并带有注册的方法集。单元数组、结构体和对象都不含有虚数部分。

mxArray 中有一个元素用来表示数组大小，它由一个数值向量构成。向量中的每个整

数都对应某一维的长度。如果向量元素的个数大于 2，那么 mxArray 就是一个多维数组。如果向量中的任一整数为零，那么 mxArray 数组将被认为是一个空数组。用户可以使用另外一个 mxArray 结构中的元素将非复数数组标记为逻辑数组。

稀疏矩阵通常由两个数值向量及指向它们的指针构成，这些向量包含了稀疏矩阵的非零元素。另外，稀疏矩阵中还包含 nzmax、ir 和 jc 参数。其中，ir 是指向矩阵中相应非零元素的行下标的指针，jc 则是指向列下标的指针，nzmax 则保存了稀疏矩阵中非零元素的最大数量。

MX 函数

Matlab 提供了不下 100 个函数和子程序用于访问和处理从 C 或 FORTRAN 程序中获得的 mxArray 结构。例如，逻辑系列子程序（mxIs...）通常返回 1（结果为真时）或 0（结果为假时）。创建系列函数（mxCreate...）通常有两个版本：mxCreate...Matrix 版本和 mxCreate...Array 版本，其中，mxCreate...Matrix 版本用于创建二维 mxArray 结构，mxCreate...Array 版本则用于创建多维 mxArray 结构。只能用于 FORTRAN 的 mxCopyPtrTo...系列函数用于把数据从 mxArray 结构拷贝到 FORTRAN 数组中，相应的，mxCopy...ToPtr 系列函数用于将 FORTRAN 数组中的数据拷贝到 mxArray 结构中。这两个系列的函数通常用来将 mxArray 结构拷贝到 FORTRAN 数组中，将数据发送到子程序中，以及将 FORTRAN 计算的结果返回到 mxArray 结构中。本章后面的例子将着重介绍上述函数和子程序的用法，有关 MX 函数和子程序的完整列表和信息（包括这些函数的参数、返回值、适用范围等）请读者参阅相关的帮助文档。

34.2　在 Matlab 中调用 C 或 FORTRAN

MEX 文件是已经编译好的 C 或 FORTRAN 函数，它们可以像标准的 M 文件一样在 Matlab 环境中调用。对于已有的 C 函数或 FORTRAN 子程序，用户可以通过加入几行代码使其能够访问 Matlab 的数据和函数。利用 mex 命令编译修改后的 C 或 FORTRAN 代码将会生成可以在 Matlab 中调用的 MEX 文件。尽管大多数计算使用 Matlab 的 M 文件执行速度很快，效率很高，但像 For 循环这样的代码在 C 或 FORTRAN 环境中执行的效率会更高。如果用户不能通过向量化的方法来消除迭代耗时，或通过循环优化技术增强计算性能的话，那么可以试着创建一个 MEX 文件来解决。

编译后的 MEX 文件带有与操作平台相关的特定的文件扩展名。例如，在 Windows 平台下，MEX 文件的扩展名为 dll；在 Sun Solaris 平台下，MEX 文件的扩展名为 mexsol。无论哪种平台下，mexext 函数都可以返回相应的 MEX 文件的扩展名。

用户所创建的每个 MEX 文件都有一个与之相关联的 M 文件（myfunc.m）来为 MEX 函数提供帮助说明。有关使用其他体系结构和编译器的附加信息请读者参考 Matlab 的帮助文档。

初始化 MEX 环境

MEX 环境必须要经过初始化，才能访问已安装的 C 或 FORTRAN 编译器。Matlab 可

以支持多种编译器，甚至包括一些官方免费提供的编译器，如 GCC 编译器（包括对 FORTRAN 的支持）。读者可以从因特网上的许多站点下载基于多种系统平台的 GCC 编译器，包括 GCC 的官方主页 http://gcc.gnu.org/。用户还可以选择在 PC 平台上使用 Matlab 自带的 LCC 编译器。另外，Matlab 还支持第三方提供的标准的 ANSI C 编译器和 FORTRAN 编译器，例如，PC 平台上的 Microsoft 和 Borland 编译器，Linux 和 BSD 平台上的 GCC 编译器，以及 Compaq、Sun 和 HP 服务器上由供应厂商提供的可选的 ANSI C 编译器。

在默认情况下，有些编译器不能屏蔽浮点溢出异常。由于 Matlab 数组中允许包含一些无穷量值，如 Inf 和 NaN，因此，如果 Matlab 没有将它们屏蔽起来，那么在使用其他编译器时将可能造成浮点溢出异常。Matlab 和编译器的帮助文档中都对如何在编译 MEX、引擎和 MAT 文件程序时屏蔽浮点异常进行了描述。

>>mex –setup 命令将为编译器和计算机平台（操作系统和体系结构）选择适宜的初始化文件或选项文件。该选项文件将为计算机平台和编译器设置某些环境变量，并指定相应的头文件和库文件的存储位置。在 Unix、Linux 和 Macintosh 平台中，选项文件被保存在 $Matlab/bin 目录中，在 Windows PC 平台中，选项文件被保存在$Matlab\bin\win32\mexopts 目录中。

MEX 环境初始化完成后，就可以使用>>mex myprog.c 命令将 MEX 源文件 myprog.c 编译成 MEX 文件 myprog.dll（或 myprog.mexglx 等）。如果需要，用户也可以在 mex 命令行中使用选项-f 临时选择另一个不同的初始化文件。我们将在后面编译 Matlab 引擎和 MAT 程序时用到-f 选项。除-f 选项外，-v 选项可以用来列出编译器的设置，并且可以观察编译和连接时每一阶段的情景。

>>mex –help 命令将列出所有的 mex 命令行选项。mex 同时也支持其他标准的编译器选项，如：-c、-g、-D。如果用户使用了-g 选项，就可以使用调试器（例如 dbx、ddd 或 gdb）来调试 MEX 程序。mex 命令既可以作为一个函数在 Matlab 命令窗口中调用执行，也可以作为一个批处理或脚本文件在 Matlab 环境以外使用。有关 mex 命令、命令行选项、所支持的编译器和调试过程的详细内容请参考 Matlab 帮助文档。

> MEX 文件必须设计成带有可变个数的输入参数和输出参数的标准 Matlab 函数来进行操作。普通的 Matlab 函数都是通过数值而非参量来传递数据的。MEX 函数的输入参数是不能修改的，如果某个 MEX 函数计算的目的就是修改其输入参数，那么被修改的值必须作为该函数的输出返回。例如：如果 MEX 函数 inc 需要将输入变量 b 的值增大，那么应该使用如下的调用方式：b=inc(b)。

MEX 源文件的组织

MEX 源代码文件可以分为两大类：接口部分和子程序部分。接口部分（在 Matlab 的帮助文档中也称为网关例程（gateway routine））包含了所有可能用到的#include 和#define 宏指令，以及访问 Matlab 数据和函数所用的代码。子程序部分（在 Matlab 的帮助文档中也称为计算例程（computation routine））用来执行数据的实际运算。该部分可以是单独的函数或子程序代码，也可以是能够合并到接口部分的代码。

接口部分必须包含一个#include "mex.h"宏指令，以提供对 MEX 函数的支持。凡是以

mex 为前缀的函数都表示这些函数只能在 MEX 文件中使用，并且能够在 Matlab 工作区中进行运算。"mex.h"头文件还包含"matrix.h"，以及一些标准的头文件<stdio.h>，<stdlib.h>和<stddef.h>，以便可以使用 MX 函数提供对 Matlab 数据类型的支持。凡是以 mx 为前缀的函数都表示它们是对 Matlab 数据类型进行运算的函数。

接口部分通常用于与 Matlab 环境进行通信。每一个 MEX 函数都需要一个 mexFunction 语句。该语句是 MEX 函数的程序入口点，相当于 C 程序中的 main 函数。mexFunction 函数的定义语法如下：

```
void mexFunction(int nlhs,        mxArray *plhs[],
                 int nrhs,  const mxArray *prhs[])
```

其中，nlhs 变量是一个整数，它包含了输出参数的个数，其值与 Matlab 的 nargout 函数返回的数值相等。同样，nrhs 代表输入参数的个数。prhs[ ]是一个指向输入参数的指针数组，例如，prhs[0]是指向第一个输入参数的指针，prhs[1]是指向第二个参数的指针，依此类推。因为所有的 Matlab 数据都是数组类型，因此所有 MEX 文件的输入和输出参数也都是 mxArray 数据类型。

在调用编译后的 MEX 函数时，prhs 参数将包含指向输入参数的指针，而 plhs 参数包含的则是空指针。所有的输出数组都需要程序员亲自创建，并将指向它们的指针放在 plhs 指针数组中。即使 nlhs 中只包含 0，即不需要任何输入参数的情况下，程序员也可以创建一个或多个输出数组。如果从命令窗口调用编译后的 MEX 函数，那么所有的输出将被赋给 Matlab 基本工作区中的 ans 变量。

MEX 函数工作区

当一个 M 文件函数被调用时，该函数将会在一个与 Matlab 基本工作区（称为调用工作区）完全不同的工作区中进行运算。基本工作区（或调用工作区）中的变量是通过数值而非参量传递给函数的，它们不受被调用函数的影响。但 evalin 和 assignin 函数除外，这两个函数可以影响和改变当前工作区以外的变量。

MEX 函数也只能在它们自己的环境中进行运算，也就是说，局部变量只在 MEX 函数体内部有效。不过，MEX 函数的运算环境和 Matlab 工作区并不完全相同。MX 函数（带有 mx 前缀的函数）用于对 Matlab 数据类型进行运算。MEX 函数（带有 mex 前缀的函数）可以在 Matlab 工作区内进行运算。例如：mexEvalString 和 mexCallMatlab 函数都可以在调用工作区中计算它们的字符串参数；mexGetVariable 函数可以从指定的 Matlab 工作区（'base'、'caller'或'global'）中将一个变量拷贝到 mxArray 中；mexGetVariablePtr 可以获得一个指向 Matlab 变量的只读指针；mexPutVariable 可以将 mxArray 结构拷贝到指定的 Matlab 工作区中。其他的 MEX 函数也都能在调用工作区中进行运算。

另一个具有不同用法的函数是用于打印格式化字符串的 mexPrintf 函数。在 MEX 文件中最好不要使用 C 中标准的 printf 函数，而应该使用 mexPrintf 函数。mexPrintf 函数将调用基本工作区中的 printf 函数，然后在命令窗口中进行打印（如果使用了日志文件，也会在日志文件中进行打印）。

下表列出了能够在 C 程序和 FORTRAN 程序中使用的 MEX 函数：

函数	C	F	功能
mexAtExit	C	F	注册一个在 MEX 文件被清除时需要调用的函数
mexCallMATLAB	C	F	调用一个 Matlab 函数、M 文件或 MEX 文件
mexFunction	C	F	MEX 文件的入口点函数
mexFunctionName	C	F	当前 MEX 函数的函数名
mexGetVariable	C	F	从另一个工作区中获取一个变量的拷贝
mexGetVariablePtr	C	F	从另一个工作区中获取指向一个变量的只读指针
mexPutVariable	C	F	将一个 mxArray 拷贝到 Matlab 工作区中
mexEvalString	C	F	在调用工作区中执行一条 Matlab 命令
mexErrMsgTxt	C	F	发出一条错误信息，并返回到 Matlab 中
mexErrMsgIdAndTxt	C	F	发出一条错误信息和一个标识符，并返回到 Matlab 中
mexWarnMsgTxt	C	F	发出一条告警信息
mexWarnMsgIdAndTxt	C	F	发出一条告警信息和一个标识符
mexLock	C	F	锁定 MEX 文件，这样就不能将其从内存中清除
mexUnlock	C	F	为 MEX 文件解锁，这样就可以从内存中清除它
mexIsLocked	C	F	如果 MEX 文件被锁定，则返回 True
mexIsGlobal	C	F	如果 mxArray 是全局范围的，则返回 True
mexPrintf	C		ANSI C 的 printf 类型的输出例程
mexPrintf		F	FORTRAN 中的输出例程，但不支持可选参数，如格式化字符串等
mexSetTrapFlag	C	F	控制 mexCallMatlab 函数对错误的响应
mexMakeArrayPersistent	C	F	使 mxArray 在 MEX 文件完成后仍持续存在
mexMakeMemoryPersistent	C	F	使由 mxMalloc 和 mxCalloc 分配的内存持续存在
mexGet	C		获得句柄图形属性的值
mexSet	C		设置句柄图形属性的值

如果与 Matlab 6 相比较，可以发现，Matlab 7 用一个 mexGetVariable 函数代替了 Matlab 6 中的 mexGetArray、mexGetMatrix、mexGetFull 3 个函数；同样，用一个 mexPutVariable 函数代替了 Matlab 6 中的 mexPutArray、mexPutMatrix、mexPutFull 3 个函数。另外，指针函数（mexGetVariablePtr）也有同样的替换现象。

Matlab 7 已不再保留和支持 mxGetName 函数（该函数用于返回一个 mxArray 的 Matlab 名称（有点类似于 Matlab 的内置函数 inputname））。为了做到向下兼容，用户也可以在 mex 脚本中使用-V5 选项，以便支持 Matlab 的 MEX 文件，但这一功能可能会在以后的版本中消失。没有了 mxGetName 函数，将一个 Matlab 变量的名称传递给变量的惟一方法是将该变量名以字符串的形式传递给 MEX 函数的参数，然后使用 mexGetVariable 函数获取变量中的数据（此时可以直接使用变量的名称）。

MEX 示例之一：fact

本例将创建一个用来计算阶乘的 MEX 函数，为了简化功能，本例略去了注释和错误检查。下面是本例的 C 函数代码：

```
/*
 * fact.c - returns the factorial of a nonnegative integer.
 *
 *   MATLAB usage:  p=fact(n)
 *
 * Mastering MATLAB 7 C MEX Example 1
*/

#include  "mex.h"

void mexFunction( int nlhs,        mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
    double n, j, *p;
    int i;

    n=mxGetScalar(prhs[0]);
    plhs[0]=mxCreateDoubleMatrix(1.1,mxREAL);
    p=mxGetPr(plhs[0]);

    j=1.0;
    for (i=n;i>1;i--)
    j=j*i;
    *p=j;
}
```

在上面的 MEX 函数中，首先利用一条宏指令将 mex.h 头文件包含进来，接着声明了 mexFunction 函数和所需要的参数。随后声明了几个变量，并通过 mxGetScalar 函数获得输入参数的值，然后使用 mxCreateDoubleMatrix 函数创建了一个 1×1 的双精度实数输出矩阵，并使用 mxGetPr 函数获得了一个 C 指针。最后执行阶乘运算，并将运算结果赋给输出矩阵。

fact.c 文件建立后，就可以在 Matlab 命令窗口中执行>>mex fact.c 命令，来创建一个可以在 Matlab 中调用的 MEX 文件。所生成的 MEX 文件的文件名由计算机平台决定：在 Windows PC 平台上为 fact.dll；在 Linux 平台上则为 fact.mexglx。如果 C 源文件中有错误，那么编译器将显示错误信息，以提示用户对源代码进行必要的修正。如果发现错误，则 MEX 输出文件将不会被创建。

运行编译好的 MEX 文件将产生如下的输出：

```
>> x =5
x =
    5
>> y = fact(x)
y =
    120
```

```
>> which fact
/home/work/matlab/fact.mexglx
```

上例仅仅使用了一些非常普通的 MX 和 MEX 函数，另外还有大量的函数没有用到。有关 Matlab 接口函数及其参数和返回值（即 Matlab 应用程序编程接口）的完整内容可以参见 HTML 和 PDF 格式的帮助文件。

MEX 示例之二：mycalc

该例将向大家介绍更多的构成 MEX 函数的元素。例如，该例中将包含错误检查，并使用了一个子程序来执行针对数组元素的计算。该例中使用的方法可以将一个已有的 C 子程序转化为 MEX 文件，要完成这一操作，用户只需在原来的 C 源代码中添加接口部分代码，用一个宏指令将 mex.h 头文件包含进来，然后进行编译即可。

下面介绍的 MEX 文件 mycalc.c 将接受一个二维双精度数组，然后对数组中的元素进行计算，并将计算结果返回到与输入数组相同大小的一个数组中。程序的接口部分主要用于进行一些错误检查，决定输入数组的维数，创建指向输入和输出数组的 C 指针，以及调用子程序进行运算等。下面是本例的 C 函数代码：

```
/*
 * mycalc.c - calculates x^2-x+1/x for each element of an array.
 *
 *   MATLAB usage: p=mycalc(n)
 *
 * Mastering MATLAB 7 C MEX Example 2
 *       single 2-D real numeric array input,
 *       single array output.
 */

#include "mex.h"

/* This is the original subroutine that performs the calculation. */
static void mycalc( double p[], double n[], int r, int c)
{
   int i;
   for (i=0;i<r*c;i++)
      p[i]=n[i]*n[i]-n[i]+1.0/n[i];
}

/* This is the interface to MATLAB data types and arguments. */
void mexFunction(int nlhs,          mxArray *plhs[],
                 int nrhs,const     mxArray *prhs[] )
{
   double *p, *n;
   int r, c ;

   /* Do some error checking. */
   if (nrhs != 1)
      mexErrMsgTxt("One input argument required.");
   else if (nlhs > 1)
      mexErrMsgTxt("Too many output arguments.");
```

```
        else if (!mxIsNumeric(prhs[0]))
           mexErrMsgTxt("Input must be numeric.");
        else if (mxIsComplex(prhs[0]))
           mexErrMsgTxt("Input must be real.");
        else if (mxGetNumberOfDimensions(prhs[0]) > 2)
           mexErrMsgTxt("N-Dimensional arrays are not supported.");
        /* Get the input array dimensions. */
        r=mxGetM(prhs[0]);
        c=mxGetN(prhs[0]);
        /* Create a matrix for the return argument */
        plhs[0]=mxCreateDoubleMatrix(r,c,mxREAL);
        /* Assign pointers to the parameters. */
        p=mxGetPr(plhs[0]);
        n=mxGetPr(prhs[0]);
        /* Do the actual calculation in a subroutine. */
        mycalc(p,n.r,c);
    }
```

在 MEX C 源文件中首先给出子程序声明，可以避免前向引用造成的麻烦，因为如果需要引用子程序，就必须在引用之前对子程序进行声明。mexErrMsgTxt 函数与 Matlab 的 error 函数功能相同。执行该函数后，错误信息将被打印在 Matlab 命令窗口中，同时退出 MEX 函数。请注意：C 语言中的字符串标识与 Matlab 不同，需要使用双引号（"）来分隔。mexWarnMsgTxt 函数大致等效于 Matlab 的 warning 函数，它在不退出 MEX 文件的情况下在命令窗口中打印告警信息。另外，使用 mexWarnMsgIdAndTxt 和 mexErrorMsgIdAndTxt 函数可以在返回告警和错误信息的同时，还返回一个 Matlab 消息标识字符串。有告警和错误消息标识符的详细信息请读者参考相应的帮助文档。

上例中的 mxGetM 和 mxGetN 函数分别用来返回二维 mxArray 的行数和列数。其中的语句

```
plhs[0]=mxCreateDoubleMatrix(r,c,mxREAL);
```

用来创建一个 r×c 的 mxArray，其元素为 Matlab 的 double 类型实数。语句中使用了 mxCOMPLEX 标志，而没有使用 mxREAL 标志，表明需要创建一个复数数组。例子中的 mxGetPr 函数用于返回指向 mxArray 数据的 C 指针。通常，mxGetPr 函数返回指向 mxArray 的实部元素的指针，要想返回指向虚部元素的指针，需要使用 mxGetPi 函数。上例在最后一步调用了 mycalc 子程序，用以完成指定的运算。

MEX 示例之三：count

本例将向大家演示如何使 MEX 文件支持多维数组。同时，本例还使用了#define 宏指令来简化源代码，并将 Matlab 帮助文本包含在生成的 MEX 文件 count.c 中。下面是本例的 C 函数代码：

```
/*
 * count.c - count occurrences of values in an array.
 *
 *    MATLAB usage:  c=count(a,b,tol)
 *
 * Mastering MATLAB 7 C MEX Example 3
 */

#include <math.h>
#include "mex.h"

/* Define some variables to make life easier. */
#define A   prhs[0] /* Pointer to first right-hand-side argument  */
#define B   prhs[1] /* Pointer to second right-hand-side argument */
#define TOL prhs[2] /* Pointer to third right-hand-side argument  */
#define C   plhs[0] /* Pointer to first left-hand-side argument   */

void mexFunction( int nlhs,          mxArray *plhs[],
                  int nrhs, const    mxArray *prhs[] )
{
    int i, j, sizea, sizeb;
    double tol, vcount;
    double *a; *b, *c;

    /* Do some error checking */
    if (nrhs < 2)
        mexErrMsgTxt("Missing input arguments.");
    else if (nrhs > 3)
        mexErrMsgTxt("Too many input arguments.");
    else if (nlhs > 1)
        mexErrMsgTxt("Too many output arguments.");

    /* Get tolerance value if supplied, otherwise use EPS. */
    if (nrhs == 3) {
        if (!mxIsNumeric(TOL) || mxIsComplex(TOL) ||
            mxGetNumberOfElements(TOL) != 1 )
            mexErrMsgTxt("TOL must be a real numeric scalar.");
        tol=mxGetScalar(TOL);
        if (tol < mxGetEps())
            mexErrMsgTxt("TOL must be a positive value.");
    }
    else
        tol=mxGetEps();

/* Make sure input arrays are noncomplex numeric arrays. */
if (!mxIsNumeric(A) ||  !mxIsNumeric(B) ||
     mxIsComplex(A) || mxIsComplex(B))
  mexErrMsgTxt("Input arguments must be real of type double.");

/* Create the output mxArray the same size as A */
C=mxCreateNumericArray(mxGetNumberOfDimensions(A),
        mxGetDimensions(A),mxDOUBLE_CLASS,mxREAL);
```

```
    /* Get the number of elements in A and B and create pointers */
    /* to the input and output arrays. */
    sizea=mxGetNumberOfElements(A);
    sizeb=mxGetNumberOfElements(B);
    a=(double *) mxGetPr(A);
    b=(double *) mxGetPr(B);
    c=(double *) mxGetPr(C);

    /* Cycle through the elements of the arrays and count values */
    for (i=0;i<sizea;i++) {
        vcount=0.0;
        for (j=0;j<sizeb;j++)
        if ((fabs(a[i]-b[j])) <= tol)
               vcount++;
            c[i]=vcount;
            }
    }
```

上例除了包含 mex.h 头文件外，还包含了标准的 math.h 头文件，这是因为在计算过程中用到了 fabs 函数。此外，还使用了 4 个#define 宏指令进行变量定义，这样可以在编写程序时减少代码输入，并且可以提高代码的可读性。上例中的代码

```
C=mxCreateNumericArray(mxGetNumberOfDimensions(A),
        mxGetDimensions(A),mxDOUBLE_CLASS,mxREAL);
```

是创建 *n* 维 mxArray 的一个 MX 函数实例。大多数 mxArray 创建函数都有两种不同的形式：二维形式和多维形式，其中二维形式如下：

```
C=mxCreateNumericMatrix(mxGetM(A),mxGetN(A),mxDOUBLE_CLASS,mxREAL);
```

在二维形式的 mxArray 创建函数（mxCreate...Matrix）中，需要首先通过两个独立的参数给出行数和列数，随后是其他参数（如本例中使用了数值类型标志和实数/复数标志）。

对于多维形式的 mxArray 创建函数（mxCreate...Array），需要首先给出维的个数和一个保存各维维数的向量，之后是其他参数。例如，

```
plhs[0]=mxCreateCellMatrix(3,4);
```

将创建一个 3×4 的空单元 mxArray（二维），而

```
plhs[0]=mxCreateCellArray(3,{2,3,4});
```

将创建一个 3 维的 2×3×4 的空单元 mxArray。

上例中的 mxGetNumberOfElements 函数将返回 mxArray 的元素总个数。因为子程序使用单个下标来索引要处理的数组元素，所以需要将此元素总个数传递给子函数，而非数组的维数。

最后，我们需要创建一个 M 文件，用以保存 count 函数的 Matlab 帮助文本。该文件需要使用与 MEX 函数相同的名称命名，即 count.m，并且需要存储在和 MEX 文件相同的目录中。count.m 的代码如下：

```
function c=count(a,b,tol)
%COUNT Count Occurrences of Values in an Array.
% COUNT(A,B) returns an array the same size as A whose i-th element
% contains the number of times A(i) appears in the array B.
% A and B must be Real arrays.
%
% COUNT implements the following:
%           c = zeros(size(A));
%           for i=1:prod(size(A))
%              c(i)=sum(A(i)==B);
%           end
```

提示：当 Matlab 发现 MEX 文件和 M 文件在同一目录中，并且具有相同的文件名时，就会调用 MEX 文件来执行函数，同时使用 M 文件来提供帮助文本。

MEX 示例之四：mmcellstr

本例将向大家介绍如何在 MEX 文件中使用字符串和单元数组。本例中的 mmcellstr 函数实际上是 Matlab 的 cellstr 函数的一个 C 语言实现。mmcellstr 函数根据一个字符数组创建一个字符串单元数组，其中字符数组的每一行都被放置在单元数组的一个独立单元中。

本例还向读者展示了一些新的技术和手段，包括：在 MEX 文件中分配内存、处理空数组和复制数组等。下面是本例的 C 函数代码：

```
/*
 * mmcellstr.c - Create a cell array of strings from a 2-D character array.
 *
 * MATLAB usage: c=mmcellstr(s)
 *
 * Mastering MATLAB 7 C MEX Example 4
 */

#include "mex.h"

void mexFunction( int nlhs,       mxArray *plhs[],
                  int nrhs, const  mxArray *prhs[] )
{
    int m, n, i, j;
    char *buf;
    char **line;

/* Do some error checking */
if (nrhs < 1)
    mexErrMsgTxt("Missing input argument.");
else if (nrhs > 1)
    mexErrMsgTxt("Too many input arguments.");
else if (mxGetNumberOfDimensions(prhs[0]) != 2)
    mexErrMsgTxt("Input must be 2-D.");
else if (nlhs > 1)
    mexErrMsgTxt("Too many output arguments.");
```

```
/****** Start of Region 1 ******/
    /* If the input is already a cell array, duplicate it. */
    if (mxIsCell(prhs[0]))
        if (mxIsChar(mxGetCell(prhs[0],0))) {
            plhs[0]=mxCreateCellMatrix(mxGetM(prhs[0]),
                    mxGetN(prhs[0]));
            plhs[0]=mxDuplicateArray(prhs[0]);
            return;
        }
/****** End of Region 1 ******/

    /* Make sure the input is a character array. */
    if (!mxIsChar(prhs[0]))
        mexErrMsgTxt("Input must be a character array.");

    /* handle the empty input case. */
    if (mxIsEmpty(prhs[0])) {
        plhs[0]=mxCreateCellMatrix(1, 1) ;
        mxSetCell(plhs[0],0,mxDuplicateArray(prhs[0]));
        return;
    }

    /* Determine the dimensions of the input structure array */
    m=mxGetM(prhs[0]);
    n=mxGetN(prhs[0]);

/****** Start of Region 2 ******/
    /* Stuff the input into a string buffer. */
    /* If mxArrayToString fails to allocate the buffer, the MEX file
terminates. */
    buf=mxArrayToString(prhs[0]);

    /* Create line buffers for the individual strings. */
    line=(char **)mxCalloc(m,sizeof(char*));
    for (i=0;i<m;i++)
        line[i]=mxCalloc(n+1,sizeof(char));
/****** End of Region 2 ******/

    /* Parse the buffer into individual lines. */
    for (j=0;j>n;j++)
        for (i=0;i<m;i++)
            line[i][j]=buf[i+m*j];

    /* Free the string buffer and create the output cell array. */
    mxFree(buf);
    plhs[0]=mxCreateCellMatrix(m, 1);

    for (i=0;i<m;i++) {
        /* For each line, remove trailing blanks... */
        j=n;
        while (--j >= 0)
        if (line[i][j] != ' ' )
            break;
```

```
        line[i][j+1]='\0';

        /* insert the line into the output array... */
        mxSetCell(plhs[0],i,mxCreateString(line[i]));

        /* and free the line buffer memory. */
        mxFree(line[i]);
    }
    mxFree(line);
}
```

上例中，在文件头和函数定义之后，定义了一些变量。在这些变量中，buf 是一个指向字符数组（字符串缓冲区）的指针，line 是一个指向字符指针数组的指针。函数的后续部分将为这些指针指定的缓冲区分配内存。变量定义之后进行了一些错误检查，然后程序进入到 Region 1，为了仔细研究，我们将该段代码重写如下：

```
/****** Start of Region 1 ******/
    /* If the input is already a cell array,duplicate it.*/
    if (mxIsCell(prhs[0]))
        if (mxIsChar(mxGetCell(prhs[0],0))){
plhs[0]=mxCreateCellMatrix(mxGetM(prhs[0]),mxGetN(prhs[0]));
        plhs[0]=mxDuplicateArray(prhs[0]);
        return;
    }
/****** End of Region 1 ******/
```

上述代码要求输入参数必须是字符串单元数组。如果满足这一要求，单元数组将被复制然后传递给输出参数。在上面的代码段中，首先检查输入参数是否是一个单元数组（利用 mxIsCell 函数），如果是，再检查单元数组中内容所属的类。其中，mxGetCell(prhs[0],0) 用于获取输入数组的单元 0 中的内容，mxIsChar 函数用于判断获取的元素是否包含字符数据。第五行代码：

```
plhs[0]=mxCreateCellMatrix(mxGetM(prhs[0]),mxGetN(prhs[0]));
```

将创建一个和输入数组维数相同的输出数组。然后在第六行代码中使用 mxDuplicateArray 函数将输入的 mxArray 完全拷贝到输出的 mxArray 中。完全拷贝指拷贝数组中所有级别的数据，它在本质上是一种递归拷贝。第六行代码：

```
plhs[0]=mxDuplicateArray(prhs[0]);
```

将输入数组中的所有内容都拷贝到输出数组中，然后返回到调用工作区。

现在，我们已经得到了参数的正确类型和数量。在下面的代码部分，我们将获得输入字符串数组的维数，然后为字符串分配大小合适的缓冲区，并使用 mxArrayToString 函数将字符数组拷贝到新的缓冲区中。注意：字符缓冲区的大小将比字符数组的长度大 1，这是因为 mxArrayToString 函数把字符串数组转换成了 C 类型的以 NULL 结尾的数组。然后，我们为输入数组的每一行分配一个行缓冲区，行缓冲区的大小等于字符数组的宽度加上一个 C 语言中的字符串结束符号'\0'。执行上述操作的代码如下所示：

```
/****** Start of Region 2 *******/
/* Stuff the input into a string buffer. */
/* If mxArray To String fails to allocate the buffer, the MEX file
terminates. */
buf=mxArrayToString(prhs[0]);

/* Create line buffers for the individual strings. */
line=(char **)mxCalloc(m,sizeof(char*));
for (i=0;i<m;i++)
     line[i]=mxCalloc(n+1,sizeof(char));
/****** End of Region 2 *******/
```

在 MEX 文件中，最好都使用 mxCalloc（或 mxMalloc）来分配内存，并使用 mxFree 将已分配的内存释放到堆栈中。上述函数将在 Matlab 内存管理器中注册内存的分配和释放，内存管理器将在函数退出时释放所有已分配的内存。下面的代码演示了 mxFree 函数的具体用法：

```
/* Parse the buffer into individual lines. */
    for (j=0;j<n;j++)
        for(i=0;i<m;i++)
            line[i][j]=buf[i+m*j];

    /* Free the string buffer and create the output cell array.*/
    mxFree(buf);
    plhs[0]=mxCreateCellMatrix(m,1);

    for (i=0;i<m;i++){
        /* for each line, remove trailing blanks... */
        j=n;
        while(--j>=0)
            if(line[i][j] != '')
                break;
        line[i][j+1]='\0';

        /* insert the line into the output array... */
        mxSetCell(plhs[0],i,mxCreateString(line[i]));

        /* and free the line buffer memory.*/
        mxFree(line[i]);
    }
    mxFree(line);
}
```

因为 Matlab 数组是以列存储的，所以缓冲区将按列顺序保存输入数组的元素。这一点与使用 Matlab 冒号运算符（buf=chararray(:)）相似。上述代码的后半部分从 buff 中提取正确的元素保存到行缓冲区中，释放字符串缓冲区内存，最后创建输出单元数组。然后，逐行对输入数组进行处理。处理完缓冲区最后一个非空字符后，插入 C 语言的字符串结束符号（'\0'），然后复制该行，并使用 mxCreateString 函数将其由 C 字符串转换成 Matlab 字符串，并使用 mxSetCell 函数将其插入到输出单元数组中。最后，使用 mxFree 函数释放该行

的缓冲区内存，然后处理下一行。所有的行都处理完后，再最后一次调用 mxFree 函数释放行数组指针。

在 Windows PC 上运行

前面的例子也可以使用 Matlab 提供的 LCC 编译器在 Windows PC 上进行编译。虽然 LCC 编译器提供的功能不多，但它仍能够生成有效的 MEX 文件。在 Windows PC 上编译 MEX 文件，可以使用下面的命令：

```
>> mex myprog.c
```

上述命令将在当前目录下生成一个 MEX 文件——myprog.dll。在 mex 命令行后加入-v 参数，可以打印编译器的设置，并显示编译和连接状态。用户可以使用 mex –setup 命令为 LCC 编译器选择选项文件。使用-f 选项可以指定某一次编译时所采用的不同选项文件。各类编译器所需的选项文件都被保存在$Matlab\bin\win32\mexopts 目录下。另外，Matlab 帮助文档中提供了有关配置编译器和调试 MEX 程序的更详细的内容。

FORTRAN MEX 文件

在 Matlab 7 中，FORTRAN MEX 文件可以创建任何有效的 Matlab 数据类型。需要注意的是，FORTRAN 源代码是不区分大小写的，因此 MXCREATEFULL、mxcreatefull 和 mxCreateFull 代表同一函数。在本章所给的 FORTRAN 例子中，将继续使用大小写混用的写法，这样可以使 FORTRAN 代码更易于阅读。

由于目前仍有很少的 FORTRAN 编译器支持新的数据类型，因此 Matlab 将针对每个输入和输出变量向 FORTRAN 程序传递一个特殊的 integer 类型的标识符，又称为指针。FORTRAN MEX 子程序可以使用这些指针以 FORTRAN 支持的数据类型从 Matlab 数组中获取数据。在几乎所有的操作系统平台上，都必须将标识符指针声明为 integer 或 integer*4 类型，不过，64 位的处理器平台例外，在该操作系统中，必须将指针声明为 integer*8 类型。

某些 FORTRAN 编译器支持%val 结构，该结构可以用于将数值从函数获取的指针（例如：p=mxGetPr()）传递到子程序中。另外，用户也可以使用 mxCopy…程序（例如 mxCopyPtrToReal8 和 mxCopyReal8ToPtr）从 mxArray 中提取数据，将数据传送到子程序，并将结果返回到 mxArray。

在 FORTRAN 中，mexFunction 子程序的定义如下：

```
subroutine mexFunction(nlhs,plhs,nrhs,prhs)
integer plhs(*),prhs(*)
integer nlhs,nrhs
```

其中，nlhs 和 nrhs 分别保存了左侧和右侧参数的数量，plhs 和 prhs 则为指向参数本身的指针数组。

下面给出了与 C MEX 文件 fact.c 等效的 FORTRAN 文件：

```
C-------------------------------------------------------------
C       fact.f - returns the factorial of a nonnegative integer.
C
```

```
C       MATLAB usage:  p=fact(n)
C
C       Mastering MATLAB 7 FORTRAN MEX Example 1

        subroutine mexFunction(nlhs, plhs, nrhs, prhs)
C-----------------------------------------------------------
C       These are pointers: integer*4 (integer*8 on 64-bit CPUs)
C
        integer plhs(*),prhs(*)
        integer mxGetPr, mxCreateDoubleMatrix
        integer y_pr
C-----------------------------------------------------------
        integer nlhs,nrhs
        integer i
        real*8 x, y, mxGetScalar
C-----------------------------------------------------------
        x = mxGetScalar(prhs(1))
        plhs(1) = mxCreateDoubleMatrix(1, 1, 0)
        y_pr = mxGetPr(plhs(1))
C
        y = 1.0
        do 10 i=x,1,-1
            y = y * i
10      continue
C
        call mxCopyReal8ToPtr(y, y_pr, 1)
        return
        end
```

注意：和 Matlab 类似，FORTRAN 也使用从 1 开始的数组和循环索引，而 C 程序使用的是从 0 开始的索引。因此，在上述 FORTRAN 程序中，指向右侧（输入）第一个参数的指针是 prhs(1)，而在 C 程序中是 prhs[0]。

34.3 从 C 或 FORTRAN 调用 Matlab

用户除了可以在 Matlab 中调用 C 或 FORTRAN 程序外，也可以在一些大型的 C 或 FORTRAN 程序中调用 Matlab 来执行后台运算。Matlab 的这一功能被称为 Matlab 引擎。

什么是 Matlab 引擎

Matlab 引擎由一个通信库和一个小型可链接例程（Linkable Routines）集组成。可链接例程集可以将 Matlab 作为服务器过程调用，而无需链接所有的 Matlab 组件。使用 Matlab 引擎可以完成如下操作：①启动一个 Matlab 过程。②向 Matlab 传递数据。③执行 Matlab 命令。④捕获 Matlab 普通命令窗口的输出。⑤将数据从 Matlab 传递到用户程序。⑥关闭 Matlab 过程。利用 Matlab 引擎，以上这些操作都可以在 C 或 FORTRAN 环境中执行。

Matlab 引擎的工作过程

Matlab 引擎是一个后台工作过程，它与当前运行的所有交互式 Matlab 过程相互独立，也不会干扰任何用户运行的 Matlab 过程。在引擎过程启动时，会创建一个新的 Matlab 实例。此时计算机中任何要求访问 Matlab 引擎过程的程序都可以共享此过程。C 语言中通常保留的是一个独占的引擎过程（即一个引擎过程只能由一个 C 实例使用），而在 FORTRAN 语言中，引擎过程通常都是共享的。

在 Unix 操作系统中，Matlab 引擎使用管道（pipe）与 C 或 FORTRAN 程序交互；而在 Windows 操作系统中，则使用组件对象模型（COM）与 C 或 FORTRAN 程序交互。在 Unix 操作系统中，用户还可以将一个远端计算机指定为 Matlab 引擎的主机。具体的语法信息请参考 engOpen 函数。

在 Matlab 中，凡是带 eng 前缀的函数都是 Matlab 引擎函数。下表给出了 C 和 FORTRAN 语言可以使用的 Matlab 引擎函数：

函数	C	F	功能
engOpen	C	F	启动或共享一个 Matlab 引擎实例
engOpenSingleUse	C		启动一个独占（非共享）的 Matlab 引擎周期
engPutVariable	C	F	将一个 Matlab 数组（mxArray）发送到 Matlab 引擎
engGetVariable	C	F	从 Matlab 引擎中获取一个 Matlab 数组（mxArray）
engOutputBuffer	C	F	创建一个存储 Matlab 文本输出的缓冲区
engEvalString	C	F	在 Matlab 引擎中执行一条 Matlab 命令
engGetVisible	C		获取一个 Matlab 引擎周期的可视化属性设置。一个可视的引擎周期将以窗口的形式在 PC 桌面运行，并可以和用户进行交互；而一个不可视的引擎周期则在后台运行，不能和用户交互
engSetVisible	C		设置一个 Matlab 引擎周期的可视化属性
engClose	C	F	关闭 Matlab 引擎

引擎程序结构

要使用 Matlab 引擎，第一步首先要使用 engOpen 或 engOpenSingleUse 函数打开一个引擎周期。然后，使用恰当的 MX 函数（例如 mxCreateDoubleMatrix 函数）创建所需的 mxArray 结构。使用 engPutVariable 函数将 mxArray 移植到 Matlab 环境中。如果需要，使用 engOutputBuffer 函数创建一个捕捉命令窗口输出的缓冲区，然后使用 engEvalString 执行捕捉到的 Matlab 命令。使用 engGetVariable 函数从 Matlab 环境中重新获取一个 mxArray 输出，并继续在 C 或 FORTRAN 程序中处理。利用 Matlab 引擎完成任务后，可以使用 engClose 函数来结束引擎过程，并使用 mxDestroyArray 函数释放分配给 mxArray 的内存。

引擎程序描述

engOpen 函数在启动一个 Matlab 引擎时，还返回一个惟一的“引擎 ID”，该引擎 ID 可以用来对此引擎进行寻址。如果发生错误，该函数将返回 NULL。如果 engOpen 的参数是 NULL（'\0'），那么将使用 matlab 命令在本地计算机上启动引擎。在 Unix 或 Linux 计算机

上，engOpen 的参数可以是一个主机名。在这种情况下，将使用 rsh 命令在远程主机上启动引擎。用户可以通过设置 DISPLAY 环境变量，使所有由引擎生成的图像（例如 plot 命令的结果）在本地计算机上显示，而不在运行引擎程序的计算机上显示。另外还可以通过设置适当的权限，允许进行远程执行，并允许远程计算机执行的结果在本地计算机上显示。如果 engOpen 的参数不是主机名，而是其他符号（例如一个包含空格、制表符或非数字非字母的字符的字符串），那么该字符串将被逐字执行以启动引擎过程。用户可以利用这一便利功能来自定义引擎过程。例如，可以使用 ssh 对远程通信过程进行加密。在 Windows PC 平台上，engOpen 的参数必须是 NULL，因为该平台不支持远程执行。此时，响应引擎请求的任务由安装在本地 PC 上的 Matlab 应用程序完成。

用户所需要的所有 Matlab 变量都必须在引擎环境中创建。创建 Matlab 变量的方法有两种：一种方法是使用 engEvalString 函数来执行一条带有返回值的 Matlab 命令，例如 y=22*pi:pi/25:2*pi，从而得到 Matlab 变量 y；另一种方法是在引擎程序中创建一个 Matlab 数组，然后将所得到的 mxArray 结果传递给 Matlab 引擎。例如，如果在 C 中给定一个 1×10 的双精度数组 dataset，则下面的代码将在用户的计算机上打开 C 与 Matlab 之间的连接，创建一个名为 mydata 的 mxArray 结构，然后将 mxArray 移植到 Matlab 环境中，并将一个名为 newdata 的 Matlab 变量指定给 mydata 数组，然后将 mydata 数组发送到 Matlab 引擎，并在引擎程序中对数组的各元素求平方，最后将结果输出到 Matlab 变量 sqdata：

```
Engine *mat
mxArray *mydata = NULL;

mat = engOpen(NULL);
mydata = mxCreateDoubleMatrix(1, 10, mxREAL);
memcpy((void *)mxGetPr(mydata),(void *)dataset,sizeof(dataset));
engPutVariable(mat,"newdata",mydata);
engEvalString("sqdata=newdata.^2;");
```

在上例中，为了可以更清晰地演示各自不同的用法，C 和 Matlab 中使用了不同的变量名，但在一般情况下，我们应尽量采用相同的变量名，以避免混淆。

在 Matlab 中执行完希望的运算后，需要将结果返回到用户的程序中。这一步也有两种实现方法：第一种方法是首先使用 engOutputBuffer 函数（该函数主要用来获取那些通常被丢弃的 Matlab 文本）创建一个文本缓冲区，然后使用 engEvalString 函数生成输出文本，最后对字符串缓冲区进行分析。例如，下面的代码将获得 disp(sum(sqdata))命令在 Matlab 命令窗口所显示的内容，并将显示的内容输出到字符串缓冲区 buf 中：

```
char buf[256];
engOutputBuffer(mat,buf,256);
engEvalString("disp(sum(sqdata))");
```

之后，可以对缓冲区 buf 进行分析，以提取所需要的数据，如下例所示：

```
double x;
x=atof(buf+2);
```

每次调用 engEvalString 函数，都会刷新 Matlab 输出字符串缓冲区中的数据。注意：由于输出字符串缓冲区中的前两个字符通常是 Matlab 提示符“>>”，因此，代码 x=atof(buf+2)

调用了 atof 函数来跳过这两个字符。

如果所使用的结果比较庞大，那么就需要使用第二种方法：从 Matlab 工作区中直接获取一个或多个数组。例如，下面的代码实现了用户程序对 mxArray 结构的访问：

```
mxArray *sqdata = NULL;
double *sptr;

sqdata = engGetVariable(mat,"sqdata");
sptr = mxGetPr(sqdata);
```

上例中，mxGetPr 函数用来获取一个指向 mxArray 的实数部分的 C 指针。除 mxGetPr 以外，其他 MX 函数也可以用于访问 sqdata 数组。

最后需要注意的是，在完成运算后，一定要释放 mxArray 占用的内存，并在不再需要 Matlab 引擎时及时关闭 Matlab 引擎。这些操作可以用下面的代码实现：

```
mxDestroyArray(mydata);
mxDestroyArray(sqdata);
engClose(mat);
```

编译和运行引擎程序

如果用户的程序中使用了 Matlab 引擎，就必须要包含 engine.h 头文件来提供对引擎访问函数的支持，另外还需要包含其他一些必需的头文件。engine.h 头文件内部包含了 matrix.h 来提供对 MX 函数的支持。在 Matlab 中编译引擎程序的命令是 mex，该命令在执行时需要一些选项支持，如 engopts.sh 选项文件、其他编译器选项等。例如，如果 Matlab 应用程序安装在/usr/local/matlab 目录下，那么可以使用下面的代码编译一个 C 引擎程序 myprog.c：

```
mex -f /usr/local/matlab/bin/engopts.sh myprog.c
```

该命令会在当前目录下创建一个名为 myprog 的可执行程序。要执行该程序，可以在操作系统的命令提示符后输入 myprog（DOS 系统下），或双击该可执行文件的图标（Windows 系统）。如果用户希望使用与标准的选项文件不同的选项，就需要定制一个 engopts.sh 文件的副本，然后对该副本进行修改，之后在 mex 命令使用该副本选项。

编译后的引擎程序在运行时，会加载一个或多个 Matlab 共享库。因此，用户需要通过一些设置告诉操作系统这些共享库的位置。当操作系统在标准系统目录下无法找到所有共享库时，LD_LIBRARY_PATH 环境变量（或读者所使用的其他平台上等效的变量）便提供了需要进行进一步搜索的附加目录。因此，用户可以在该环境变量的路径列表中加入自己的计算机中 Matlab 共享库所在的目录。例如，如果用户使用的是 Solaris 操作系统，Matlab 的安装目录是/opt/matlab，并使用 Bourne 作业系统（sh、bash、ksh），则可以使用下面的命令在 LD_LIBRARY_PATH 环境变量中添加 Matlab 共享库所在的目录：

```
LD_LIBRARY_PATH=/opt/matlab/extern/lib/sol2:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

如果用户的操作系统是 Linux，Matlab 的安装目录是/usr/local/matlab/，并使用 C 作业系统（csh，tcsh）和 C 的 libc2 共享库，则可以使用下面的命令在 LD_LIBRARY_PATH 环境变量中添加 Matlab 共享库所在的目录：

```
setenv LD_LIBRARY_PATH /usr/local/matlab/extern/lib/glnx86:$LD_LIBRARY_PATH
```

一般情况下，与操作系统平台相关的共享库目录（如 sol2，glnx86，sgi，sgi64，win32，hpux 等）都存储在 Matlab 的$Matlab/extern/lib 目录下。在上面的操作系统平台中，并不是所有的操作系统平台都使用 LD_LIBRARY_PATH 环境变量来存储附加共享库的位置。例如，SGI64 操作系统平台使用的环境变量是 LD_LIBRARY64_PATH 变量，HPUX 操作系统平台使用的环境变量是 SHLIB_PATH，而 Mac OS X 操作系统平台使用的环境变量是 DYLD_LIBRARY_PATH。设置环境变量的最简单的方法是在用户的作业系统的启动脚本（如.profile、.cshrc、.bashrc、.tcshrc 等）中添加相应的命令。

引擎示例

前面，我们对 Matlab 引擎程序的各独立部分进行了讲解。下面，我们通过一些实例将这些独立部分合并在一起进行讲解。本节给出的例子将以前面各章节的内容为基础，并且给出了使用 Matlab 引擎进行后台计算的完整的 C 程序。该 C 程序主要实现以下功能：①创建数据数组。②将数组发送给 Matlab。③在 Matlab 中计算所有元素的平方和。④将 Matlab 结果返回到 C 程序中。⑤在屏幕上显示结果。用户可以通过程序中的注释例解各程序段的具体意义。下面是引擎程序的完整 C 代码：

```
/*
 * sos.c - Calculate the sum of the squares of the elements of a vector.
 *
 * Mastering MATLAB 7 C Engine Example 1
 *
 */

#include <stdio.h>
#include <string.h>
#include "engine.h"
#define  BUFSIZE 256

int main()
{
    Engine *mat;
    mxArray *mydata = NULL, *sqdata = NULL;
    int i;
    double x, *myptr, *sptr;
    char buf[BUFSIZE];
        double dataset[10] = { 0.0, 1.0, 2.0, 3.0, 4.0,
                               5.0, 6.0, 7.0, 8.0, 9.0 };

    /* Start the MATLAB Engine on the local computer. */
    if (!(mat = engOpen("NULL"))) {
        fprintf(stderr, "\nCannot open connection to MATLAB!\n");
        return EXIT_FAILURE;
    }

    /* Create an mxArray and get a C pointer to the mxArray. */
    mydata = mxCreateDoubleMatrix(1, 10, mxREAL);
```

```
        myptr = mxGetPr(mydata);

        /* Copy the dataset array to the new mxArray. */
        memcpy((void *)myptr, (void *)dataset, sizeof(dataset));

        /* Pass the mxArray to the Engine and square the elements. */
        engPutVariable(mat, "newdata", mydata);
        engEvalString(mat, "sqdata = newdata.^2");

        /* Create an output buffer to capture MATLAB text output. */
        engOutputBuffer(mat, buf, BUFSIZE);

        /* Calculate the sum of the squares and save the result in x. */
        engEvalString(mat,"disp(sum(sqdata))");
        x=atof(buf+2);

        /* Retrieve the array of squares from the Engine, */
        if ((sqdata = engGetVariable(mat,"sqdata")) == NULL) {
            fprintf(stderr, "Cannot retrieve sqdata!\n\n");
            return EXIT_FAILURE;
        }

        /* and get a C pointer to the mxArray. */

        sptr = mxGetPr(sqdata);

        /* Print the results to stdout. */
        printf("\nThe inputs are:\n");
        for (i=0;i<10;i++)
            printf("%6.1f",myptr[i]);
        printf("\n\n The squares are:\n");
        for (i=0;i<10;i++)
            printf("%6.1f",sptr[i]);
        printf("\n\nThe sum of the squares is %6.1f \n\n",x);

        /* Free the mxArray memory and quit MATLAB. */
        mxDestroyArray(mydata);
        mxDestroyArray(sqdata);
        engclose(mat);

        return EXIT_SUCCESS;
    }
```

上例已在 Linux 和 Macintosh 操作系统中通过了编译和测试。

在 Windows PC 中使用时的注意事项

上面给出的例子还可以在 Windows PC 平台上使用 LCC 编译器进行编译，编译命令如下：

```
mex -f c:\matlab7\bin\win32\mexopts\lccengmatopts.bat sos.c
```

该命令将在当前目录下生成可执行文件 sos.exe。如果用户不使用 LCC 编译器，则需要在命令行中指定适当的编译器选项文件。在$Matlab\bin\win32\mexopts 目录下保存着至少 8 个选项文件，分别用于不同版本的编译器选项，如 Borland、Microsoft、Watcom 等。如果

用户对编译命令和选项文件仍不十分清楚，可以在 mex 命令中使用说明选项（-v）来查看编译器的设置和编译过程的各个步骤。

在 Windows 操作系统平台中，Matlab 引擎程序要连接$Matlab\bin\win32 目录下的 DLL 库。在 Matlab 安装过程中，会将该目录添加到默认路径中，以便于 Windows 能够找到这些库文件。当用户执行 sos.exe 文件时，会在后台（即最小化方式）运行一个 Matlab 运行周期来执行平方和运算。该运行周期不会干扰任何正在运行的其他交互式 Matlab 运行周期。

有关如何配置编程软件来编译 MEX 和引擎程序，和使用软件开发环境来调试程序的详细内容，请读者参考相应的 Matlab 帮助文档。

在 FORTRAN 中使用时的注意事项

用 FORTRAN 语言编写的 Matlab 引擎程序与用 C 语言编写的引擎程序相比大同小异。大多数引擎函数都会返回一个状态值，用于进行错误检测。另外，为了避免从文本缓冲区中提取数据，程序计算出的平方和将作为字符串数组进行打印。下面是引擎程序的完整 FORTRAN 代码：

```
C
C   sos.f - Calculate the sum of the squares of the elements of a vector.
C
C   Mastering MATLAB 7 FORTRAN Engine Example 1
C
C=================================================================
C   program main
C-----------------------------------------------------------------
C   Pointers
C
    integer engOpen, engGetVariable, mxCreateDoubleMatrix, mxGetPr
    integer mat, mydata, sqdata
C-----------------------------------------------------------------
C   Other variable declarations
C
    double precision dataset(10), sqrs(10)
    integer engPutVariable, engEvalString, engClose, engOutputBuffer
    integer temp, status
    character*256 buf
    data dataset / 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0 /
C-----------------------------------------------------------------
C   Start the MATLAB Engine on the local computer.
C
    mat = engOpen('matlab')
    if (mat .eq. 0) then
        write(6,*) 'Cannot open connection to MATLAB!'
        stop
    endif
C
C   Create an mxArray, associate a MATLAB variable name with the
C   mxArray, and copy the data into the array.
```

```
c
      mydata = mxCreateDoubleMatrix(1, 10, 0)
      call mxCopyReal8ToPtr(dataset, mxGetPr(mydata), 10)
c
c     Pass the variable mydata into the MATLAB workspace.
c
      status = engPutVariable(mat, 'newdata', mydata)
      if (status .ne. 0) then
          write(6,*) 'Cannot pass mydata to the Engine!'
          stop
      endif
c
c     Square the elements of the array.
c
      if (engEvalString(mat, 'sqdata = newdata.^2;') .ne. 0) then
          write(6, *) 'engEvalString failed'
          stop
      endif
c
c     Create an output buffer to capture MATLAB text output.
c
      if (engOutputBuffer(mat, but) .ne. 0) then
          write(6,*) 'engOutputBuffer failed'
          stop
      endif
c
c     Calculate the sum of the squares and capture the result.
c
      if (engEvalString(mat, 'disp(sum(sqdata))') .ne. 0) then
          write(6,*) 'engEvalString failed'
          stop
      endif
c
c     Retrieve the mxArray of squares from the Engine,
c     copy the data into an array of doubles, and print.
c
      sqdata = engGetVariable(mat, 'sqdata')
      call mxCopyPtrToReal8(mxGetPr(sqdata), sqrs, 10)
c
 20   format(' ', G8.3, G8.3, G8.3, G8.3, G8.3, G8.3,
      & G8.3, G8.3, G8.3, G8.3)
      print *, 'The inputs are:'
      print 20, dataset
      print *, 'The squares are:'
      print 20, sqrs
      print *,  'The sum of the squares is ', buf(3:10)
c
c     Free the mxArray memory and quit MATLAB.
c
```

```
      call mxFreeMatrix(mydata)
      call mxFreeMatrix(sqdata)
      status = engClose(mat)
C
      if (status .ne. 0) then
          write(6,*) 'engClose failed'
          stop
      endif
C
      stop
      end
```

注意：如果您的 FORTRAN 编译器使用了共享库，那么在执行引擎程序前，必须先将 Matlab 共享库的目录添加到 LD_LIBRARY_PATH（或其他等效的）环境变量中。但如果您使用的是 Windows PC，那么 Matlab 会在初始安装时自动将$Matlab\bin\win32 目录添加到默认路径中，因此 Windows 可以直接找到 Matlab 共享库（DLL）的位置，用户无需设置环境变量。

34.4　与 MAT 文件交换数据

Matlab 有多种与其他程序进行数据交换的方式，但大多数方式都需要创建一个作为"交换中介"的数据文件。选择何种方式取决于导入或导出数据的数量和格式。本节将着重介绍如何在 C 和 FORTRAN 中读写 Matlab 标准的 MAT 文件。

什么是 MAT 文件

MAT 文件是与操作系统平台无关的，这是因为 MAT 文件本身已经包含了可以识别平台差别（例如字节次序）的信息，因此，Matlab 在从 MAT 文件装载数据时会自动将数据格式转化为本地操作系统需要的形式。用户只要在程序中提供读写 MAT 文件所需的头文件和库文件，就可以在 C 和 FORTRAN 程序中使用 Matlab 提供的 MAT 文件，从而完成与 Matlab 之间的数据交换。

MAT 函数

MAT 函数通常带有 mat 前缀，主要用于对 MAT 文件进行操作。下表给出了 C 和 FORTRAN 语言中常用的 MAT 函数：

MAT 函数	C	F	作用
matOpen	C	F	打开一个 MAT 文件
matClose	C	F	关闭一个 MAT 文件
matGetDir	C	F	获取 MAT 文件中的 Matlab 数组的列表
matGetFp	C		获取一个指向 MAT 文件的 ANSI C 文件指针
matGetVariable	C	F	从 MAT 文件中读取一个 Matlab 数组
matGetVariableInfo	C	F	从 MAT 文件中装载一个 Matlab 数组头

（续表）

MAT 函数	C	F	作用
matGetNextVariable	C	F	从 MAT 文件中读取下一个 Matlab 数组
matGetNextVariableInfo	C	F	从 MAT 文件中装载下一个 Matlab 数组头
matPutVariable	C	F	将一个 Matlab 数组写入 MAT 文件
matPutVariableAsGlobal	C	F	将一个 Matlab 数组作为全局变量写入 MAT 文件
matDeleteVariable	C	F	将一个 Matlab 数组从 MAT 文件中删除

上表中，matGetVariableInfo 函数用于创建一个包含 mxArray 结构中除数据以外的其他所有信息的 mxArray 变量。matGetDir 函数用于创建一个变量名称列表并返回 MAT 文件中 Matlab 变量的数量。matGetVariable 函数用于访问变量的内容。另外，Matlab 还提供了 matGetNextVariable 和 matGetNextVariableInfo 函数用于访问 MAT 文件中的下一个变量。MatPutVariableAsGlobal 函数用于将一个 Matlab 数组作为全局变量写入 MAT 文件，也就是说，当使用 load 命令将 MAT 文件载入 Matlab 时，该变量将以全局变量的方式载入 Matlab 工作区。

MAT 程序结构

在 C 或 FORTRAN 程序中读写 MAT 文件时，仍需要使用 MX 函数来创建 Matlab 数据数组（mxArray），其创建方法和前面的 MEX 文件或引擎程序中的创建方法相同。要创建 MAT 文件，首先需要使用 matOpen 函数打开该 MAT 文件；然后需要为每个 Matlab 变量创建一个 mxArray，给 mxArray 进行恰当命名，并将 Matlab 变量移植到 mxArray 中；然后使用 matPutVariable 和 matPutVariableAsGlobal 函数将 mxArray 写入 MAT 文件中；最后使用 matClose 函数关闭 MAT 文件。要从 MAT 文件中读取数据，首先需要使用 matOpen 函数打开该 MAT 文件；如果需要变量列表，可以使用 matGetDir 函数获得一个列表；然后使用 matGetVariable 或 matGetNextVariable 函数创建 mxArray；最后使用 matClose 函数关闭 MAT 文件。

编译和运行 MAT 程序

如果用户的程序中使用了访问 MAT 文件的函数，就必须要包含 mat.h 头文件来提供对 MAT 函数的支持，另外还需要包含其他一些必需的头文件。mat.h 头文件内部包含了 matrix.h 来提供对 MX 函数的支持。在 Matlab 中编译 MAT 程序的命令是 mex，该命令在执行时需要一些选项支持，如 engopts.sh 选项文件、其他编译器选项等。例如，如果 Matlab 应用程序安装在/usr/local/matlab 目录下，那么可以使用下面的代码编译一个用 C 编写的 MAT 程序 myprog.c：

```
mex -f /usr/local/matlab/bin/matopts.sh myprog.c
```

该命令将在当前目录下创建一个名为 myprog 的可执行程序。与引擎程序一样，MAT 程序中同样需要与 UNIX PC 平台有关的共享库，并且，在运行由 mex 命令创建的可执行文件时，操作系统必须能够找到需要的共享库。

对于 Windows PC 平台，其选项设置与引擎程序相同。比如，如果用户使用 LCC 编译器编译上面的 MAT 文件 myprog.c，可以使用下面的命令：

```
mex -f c:\matlab7\bin\win32\mexopts\lccengmatopts.bat myprog.c
```

该命令将在当前目录下生成可执行文件 myprog.exe，用户可以双击运行该文件。如果用户使用的不是 LCC 编译器，那么需要在命令行中指定所需的编译器选项文件。

MAT 编程示例之一：writemat

下面给出的 C 程序将创建一个 MAT 文件，其中包含了一个字符串和一个双精度数组：

```
/*
 * writemat.c - Create a binary MAT-file.
 *
 * Mastering MATLAB 7 C MAT-file Example 1
 *
 */
#include "mat.h"

int makemat(const char *filename,
            double *data, int m, int n,
            char *mmstr)
{
MATFile *mfile;
mxArray *mdata, *mstr;

/* Open the MAT-file for writing. */
mfile = matOpen(filename, "w");
    if (mfile == NULL) {
printf("Cannot open %s for writing.\n", filename);
return(EXIT_FAILURE);
}

/* Create the mxArray to hold the numeric data.   */
/* Note that the array dimensions are reversed.   */
/* C uses row order while MATLAB uses column order. */
/* The data array will be transposed in MATLAB.   */
mdata = mxCreateDoubleMatrix(n,m,mxREAL);
mxSetName(mdata, "mydata");

/* Copy the data to the mxArray. Note that mxGetData is */
/* similar to mxGetPr but returns a void pointer while */
/* mxGetPr returns a pointer to a double.         */
memcpy((void *)(mxGetData(mdata)),(void *)data,
                m*n*sizeof(double));

/* Create the string array. */
mstr = mxCreateString(mmstr);

/* Write the mxArrays to the MAT-file. */
matPutVariable(mfile, "mydata", mdata);
```

```
    matPutVariable(mfile, "mystr", mstr);

    /* Free the mxArray memory. */
    mxDestroyArray(mdata);
    mxDestroyArray(mstr);

    /* Close the MAT-file. */
    if (matClose(mfile) != 0) {
        printf("Cannot close %s.\n",filename);
        return(EXIT_FAILURE);
    }
        return(EXIT_SUCCESS);
    }

    int main( )
    {
        int status;
        char *mmstr = "Mastering MATLAB Rocks!";
        double data[3][4] = {{  1.0,   2.0,  3.0,   4.0 },
                             {  5.5,   6.6,  7.7,   8.8 },
                             { -4.0,  -3.0, -2.0,  -1.0 }};
        status = makemat("mmtest.mat", *data, 3, 4, mmstr);
        return(status);
    }
```

该程序已经在 Linux、Macintosh 和 Windows PC 平台上编译和测试成功。上面的程序所得到的 MAT 文件可以被加载到任何一个 Matlab 执行周期中，其加载代码和运行结果如下所示：

```
>> clear all
>> load mmtest
>> whos
   Name            Size              Bytes  Class
   mydata          4×3                  96  double array
   mystr           1×23                 46  char array
Grand total is 35 elements using 142 bytes

>> mydata
mydata =
       1.0000      5.5000      -4.0000
       2.0000      6.6000      -3.0000
       3.0000      7.7000      -2.0000
       4.0000      8.8000      -1.0000

>> mystr
mystr =
Mastering MATLAB Rocks!
```

注意：在 Matlab 从 MAT 文件中装载数值数组时，需要对数值数组进行转置。这是因为 C 语言和 Matlab 存储数组的方式不同：C 语言按行进行存储，而 Matlab 按列进行存储。但在 FORTRAN 语言中，就不存在这种转置，因为 FORTRAN 和 Matlab 都是按列存储数组的。下面给出了上例的 FORTRAN 语言文件：

```
C
C   writemat.f - Create a binary MAT-file.
C
C   Mastering MATLAB 7 FORTRAN MAT-file Example 1
C
C
    program writemat
C-------------------------------------------------------------
C   Pointers.
C
    integer matOpen, mxCreateDoubleMatrix, mxCreateString
    integer matGetVariable, mxGetPr
    integer mfile, mdata, mstr
C-------------------------------------------------------------
C   Other variables
C
    integer status, matClose
    double precision dat(12)
    data dat /  1.0, 5.5, -4.0,
    &           2.0, 6.6, -3.0,
    &           3.0, 7.7, -2.0,
    &           4.0, 8.8, -1.0 /
C
C   Open MAT-file for writing.
C
    mfile = matOpen('mmtest.mat', 'w')
    if (mfile .eq. 0) then
       write(6,*) 'Can''t open ''mmtest.mat'' for writing.'
       stop
    end if
C
C   Create the mxArray to hold the numeric data.
C
    mdata = mxCreateDoubleMatrix(4,3,0)
C
C   Copy the data to the mxArray.
C
    call mxCopyReal8ToPtr(dat, mxGetPr(mdata), 12)
C
C   Create the string array.
C
    mstr = mxCreateString('Mastering MATLAB Rocks!')
C
C   Write the mxArrays to the MAT-file.
C
    call matPutVariable(mfile, 'mydata', mdata)
    call matPutVariable(mfile, 'mystr', mstr)
C
C   Free the mxArray memory.
```

```
C
      call mxFreeMatrix(mdata)
      call mxFreeMatrix(mstr)
C
C     Close the MAT-file.
C
      status = matClose(mfile)
      if (status .ne. 0) then
         write(6,*) 'Cannot close mmtest.mat'
         stop
      end if
C
      stop
      end
```

MAT 编程示例之二：whomat

该例将读入一个 MAT 文件，检验文件的内容，然后像 Matlab 的 who 和 whos 命令一样打印出文件中的变量列表。本例的 C 语言代码如下：

```
/*
 * whomat.c - Examine a binary MAT-file and print a list
 *            of the contents(like "who" or "whos").
 *
 * Mastering MATLAB 7 C MAT-file Example 2
 *
 */

#include "mat.h"
#include <string.h>

int whomat(const char *filename)
{
    MATFile *mfile;
    mxArray *marray;
    char **dir;
    char siz[25], buf[10];
    int i, j, num, nel, elsize, ndim, eltot, btot;
    const int *dims;

    /* Open the MAT-file for reading. */
    mfile = matOpen(filename, "r");
    if (mfile == NULL) {
        printf("Cannot open %s for reading.\n", filename);
        return(EXIT_FAILURE);
    }

    /* Get the directory list and print in "who" format. */
    dir = matGetDir(mfile, &num);
    if (dir == NULL) {
```

```
        printf("Error reading the directory of %s.\n", filename);
        return(EXIT_FAILURE);
    } else {
        printf("\n") ;
        printf("Variables in %s are:\n\n", filename);
        for (i=0; i<num; i++) {
            printf("%-10s",dir[i]);
            if (i>0 && i%4==0) printf("\n");
        }
    }

    /* Examine each variable and print a "whos" list. */
    eltot=btot=0;
    printf("\n\n Name          Size          Bytes  Class\n\n");
    for (i=0; i<num; i++) {
        marray=matGetVariable(mfile, dir[i]);
        if (marray == NULL) {
            printf("Cannot read file %s.\n\n", filename);
            return(EXIT_FAILURE);
        }

        /* If marray is a cell array or structure array, then  */
        /* mxGetElementSize returns the size of a pointer, not */
        /* the size of all the elements in each cell or field. */
        /* To get the correct number of bytes would require    */
        /* traversing the array and summing leaf element sizes.*/
        /* Java arrays return 0X0 array dimensions and 0 size. */

        elsize=mxGetElementSize(marray);
        btot=btot+(nel*elsize);
        nel=mxGetNumberOfElements(marray);
        eltot=eltot+nel;
        ndim=mxGetNumberOfDimensions(marray);
        dims=mxGetDimensions(marray);
        siz[0]='\0';
        for (j=0; j<ndim; j++) {
            sprintf(buf,"%d",dims[j]);
            strcat(siz,buf);
            if (j<(ndim-1))
                strcat(siz,"x");
        }
        printf(" %-12s %-12s %5d %s array\n",dir[i],
                siz,nel*elsize,mxGetClassName(marray));
        mxDestroyArray(marray);
    }
    printf("\nGrand total is %d elements using %d bytes\n\n",
            eltot,btot);

    /* Release the memory allocated for the directory. */
        mxFree(dir);

    /* Close the MAT-file. */
```

```
        if (matClose(mfile) != 0) {
            printf("Cannot close %s.\n",filename);
            return(EXIT_FAILURE);
        }
        return(EXIT_SUCCESS);
    }

    int main(int argc, char **argv)
    {
        int status;

        if (argc > 1)
            status = whomat(argv[1]);
        else{
            status = EXIT_FAILURE;
            printf("Usage: whomat <matfile>");
        }
        return(status);
    }
```

上述程序的输出与 Matlab 的 who 和 whos 命令（带有'-file'参数）的输出非常相似，只不过对于单元数组、结构体和 Java 数组将会报告出不正确的结果。如果 marray 是单元数组或结构体，那么 mxGetElementSize(marray)将返回一个指针的大小，而不是每个单元或域中所有元素的大小。为了能够得到正确的字节数，需要遍历数组并计算端点元素的尺寸之和。如果 marray 是一个 Java 数组，那么 mxGetElementSize(marray) 将返回 0，mxGetDimensions(marray) 将返回一个指向 0 向量（该向量的长度由 mxGetNumberOfDimensions(marray)确定）的指针。关于如何添加对这些类型的数据的支持，将作为练习留给读者来完成。

34.5 共享库

所谓库（Library）指可以被任何程序使用的函数集。这些函数通常被预编译在一个库文件（Library File）中。库分两种：静态库（Static Library）和共享库（Shared Library）。静态库将在用户程序编译阶段被链接，并且所引用的函数将被封装在用户的可执行文件中；共享库将在用户程序的运行阶段被链接，所引用的函数不会被封装在用户的可执行文件中。正是由于共享库不能被嵌入到用户可执行文件中，因此在用户程序运行阶段该库一定要能被找到。在同一时刻，计算机上的许多程序都可以访问同一个共享库。在 Unix 和 Linux 系统中，共享库通常被保存在标准的系统库路径（如/lib、/usr/lib、/usr/shlib 或/usr/local/lib）和 Matlab 库文件路径$MATLAB/bin/$ARCH 中。这些库文件都使用 lib 前缀和.so 后缀（Macintosh 系统则使用.dylib 后缀）。在 Windows 操作系统中，共享库就是我们常说的动态链接库（DDL），通常使用.dll 后缀。在程序运行时，所需的共享库中的函数将被装载到内存，以供应用程序调用。

利用 Matlab 提供的共享库接口，用户可以很方便地加载和卸载一个共享库，得到库中的函数列表（包括函数名、参数和返回值），在 Matlab 中调用这些函数等。只要一个共享

库具有 C 风格的参数和类型，都可以在 Matlab 的共享库接口中进行访问。

下表给出了用户可能会用到的 Matlab 共享库函数：

函数	作用
loadlibrary	将一个外部的库载入到 Matlab 中
unloadlibrary	将一个外部的库从内存中卸载
libisloaded	判断一个外部的库是否已被载入
libfunctions	返回一个外部库中的函数的信息
libfunctionsview	创建一个窗口用于显示函数的信息
calllib	调用一个外部库中的函数
libpointer	创建一个指向外部库的指针，以便使用该外部库
libstruct	创建一个类似于 C 的结构，该结构可以传递给外部的库

在大多数情况下，Matlab 都能在自身的数据类型和 C 数据类型之间进行转换。例如，如果要将一个 Matlab 结构体作为参数传递给 C 库函数，该结构体将被自动转换为 C 结构。而如果在 C 函数中，需要指针的参数处传递的是一个 Matlab 数据类型的值（注意：Matlab 中不存在指针），该值将会被自动转换为指针。除此以外，上表中的 libpointer 和 libstruct 函数允许用户显式地进行数据类型转换。更多的共享库函数和数据类型转换的信息，请读者参考相应的 Matlab 帮助文档。

34.6 串口通信

在计算机中，数据通常都是通过串口进行采集的。Matlab 提供了一个内置的接口用于管理串口，并可以直接与串口上连接的设备进行通信。Matlab 可以支持 Linux，Sun Solaris 和 Windows 操作系统中的 RS232、RS422 和 RS485 串口通信标准。另外，Matlab 开发商或第三方还提供了一些可选的工具项用于支持增强的端口通信，如多功能 I/O 板、GPIB、VESA 设备以及串口的一些附加功能等。不过，对于一般的用户而言，Matlab 提供的内置接口已经能够满足串口通信的需要了。

Matlab 的串口接口是一个面向对象的接口。使用 serial 函数可以创建一个和指定的串口相连的对象，用户可以使用 get 和 set 函数获取和设置该对象的属性。Serial 在调用时需要一个串口设备的名称，和一系列可选的属性名/属性值参数对。对于不同的操作系统，串口设备的名称也是不一样的。例如，同样是第一个串口，在 Solaris 7 操作系统中的名称是/dev/term/a 或/dev/ttya，在 i386 Linux 操作系统中则是/dev/ttyS0，在 Windows 操作系统则是 COM1。

与一个串口设备之间的通信通常都通过一系列固定的操作完成：首先，使用 serial 函数创建一个与该串口设备的接口对象，并使用 set 函数设置对象的属性；然后，使用 fopen 函数打开串口设备，并使用 fprintf 和 fscanf 函数进行读写，读写完毕后使用 fclose 函数关闭串口设备；最后使用 delete 函数删除串口对象，并使用 clear 函数清空 Matlab 工作区中的变量。

串口通信示例

下面我们举一个与血糖计进行交互式通信的例子。该血糖计需要将串口设置为9600,8,N,1，并且不需要握手协议。其中的9600,8,N,1表示串口通信的波特率为9600bps，数据位为8比特，无校验位，停止位为1比特。该血糖计不需要使用硬件握手。

首先，创建一个与第一个串口相连的串口对象，并检查它的属性，代码如下：

```
>> s = serial('/dev/ttyS0');
>> get(s)
    ByteOrder = littleEndian
    BytesAvailable = 0
    BytesAvailableFcn =
    BytesAvailableFcnCount = 48
    BytesAvailableFcnMode = terminator
    BytesToOutput = 0
    ErrorFcn =
    InputBufferSize = 512
    Name = Serial-/dev/ttyS0
    ObjectVisibility = on
    OutputBufferSize = 512
    OutputEmptyFcn =
    RecordDetail = compact
    RecordMode = overwrite
    RecordName = record.txt
    RecordStatus = off
    Status = closed
    Tag =
    Timeout = 10
    TimerFcn =
    TimerPeriod = 1
    TransferStatus = idle
    Type = serial
    UserData = []
    ValuesReceived = 0
    ValuesSent = 0

    SERIAL specific properties:
    BaudRate = 9600
    BreakInterruptFcn =
    DataBits = 8
    DataTerminalReady = on
    FlowControl = none
    Parity = none
    PinStatus = [1×1 struct]
    PinStatusFcn =
    Port = /dev/ttyS0
    ReadAsyncMode = continuous
    RequestToSend = on
    StopBits = 1
    Terminator = LF
```

对于本节的血糖计的例子，上述大部分缺省串口通信参数都是正确的。但缺省的 Terminator 参数需要修改，因为血糖计的例子需要一个回车/换行（CR/LF）作为通信结束符，而不是单纯的换行（LF）。因此，我们需要使用下面的代码进行修改：

```
>> set(s,'Terminator','CR/LF');
>> get(s,'Terminator')
ans =
CR/LF
```

下面的代码将打开串口，向血糖计发送一个命令，并使之对串口号和通信校验和做出反应：

```
>> fopen(s);
>> fprintf(s,'DM@');
>> snum = fscanf(s)
snum =
    @ "RMF99CFBV" 031D
```

下面的代码则使血糖计的读数归零，读者可以检查执行代码时血糖计的反应：

```
>> fprintf(s,'DMZ')
>> resp = fscanf(s)
resp =
    Z 005A
```

如果没有错误出现，证明上述代码是正确的。现在，用户可以关闭串口连接，删除端口对象，并将变量 s 从 Matlab 环境删除：

```
>> fclose(s)
>> delete(s)
>> clear s
```

同图形对象一样，串口对象也支持事件和回调。其中回调指特定事件发生时需要执行的函数。函数既可以通过使用函数句柄指定，也可以通过包含函数名的单元数组指定。凡是属性名中含有'Fcn'的对象属性，都可以在相应的事件发生时执行一个回调函数。串口通信中的事件通常包括到达时钟某一时刻、发生错误、中断、输出缓冲区清空、输入缓冲区中接收了数据、串口线的某一管脚（如 CD、RI、DSR 或 CTS）的状态发生了变化等等。

属性可以用来对下列操作进行监控：管脚状态控制、DTR 管脚控制、使用硬件（RTS/CTS）或软件（Xon/Xoff）执行数据流控制、自动纪录执行一个数据文件的信息。如果需要，可以将读写的数据的个数和类型、事件信息、以及所有的数据值保存在一个或多个文件中。更多的信息和详细的例子请参考相应的 Matlab 帮助文档。

34.7 源代码控制系统

对于大型的程序工程或需要多个开发软件才能完成的工程而言，通常需要使用一个源代码控制系统（SCCS）来管理程序的源代码和对源代码进行修改。在源代码控制系统中，所有的文件都将被集中在一个“中心容器”中，并可以被相应的开发软件提取出来。如果开发软件对文件进行了修改，这些文件的修改也会被 SCCS 立即核实。另外，SCCS 会在各

个开发软件的各个版本之间保持一致性，从而便于用户使用不同版本的开发软件。

许多现有的 SCCS 都可以在 Matlab 环境中进行访问。其中包括：Unix、Linux 和 Macintosh 操作系统中的 CVS、RCS、PVCS、CM Synergy 和 ClearCase，以及 Windows 操作系统中所有符合 Microsoft Common Source Control 标准的系统。除 Windows 外，大部分操作系统都支持用户定制的 SCCS 系统。

Matlab 在其编辑器窗口中为正在使用的 SCCS 系统提供了一个图形化的接口，另外还提供了一个可以在命令窗口使用的函数接口。用户可以在 Matlab 的 Preference 选项窗口（单击命令窗口中 File 菜单下的 Preference 菜单项可以打开该窗口）中选择自己喜欢的 SCCS。当 SCCS 被选定后，Matlab 编辑器中的 File 菜单下的 Source Control 菜单将变为可用，其中包括 Check In、Check Out 和 Undo Check-Out 菜单项。下表给出了用户可能会用到的一些 SCCS 函数：

函数	作用
cmopts	返回所选择的源代码控制系统的名称
checkin	将一个文件加入到源代码控制系统中
checkout	将一个文件从源代码控制系统中提取出来
undocheckout	取消前一次的 check-out 操作
customverctl	用户定制版本的控制系统模板
rcs	使用 RCS 进行版本控制操作
cvs	使用 CVS 进行版本控制操作
pvcs	使用 PVCS 进行版本控制操作
sourcesafe	使用 SourceSafe 进行版本控制操作
clearcase	使用 PVCS 进行版本控制操作

34.8 网络服务

Internet 已经成为计算机领域中一个重要的组成部分，因此，Matlab 也提供了对各种 Internet 技术的支持。例如，在 Matlab 7 中，客户端可以通过网络（局域网或 Internet）对远程程序进行调用。

首先我们先来了解一下有关网络的一些背景知识和术语。1985 年，通用标记语言标准（SGML）产生，该标准是制定各类电子文档的国际通用标准。可扩展标记语言（XML）是 SGML 的一个简化版本，它省去了 SGML 中的许多管理功能。超文本标记语言（HTML）是 SGML 的一个静态子集，它通常用于描述非常简单的报告式的文档，被广泛应用于网页开发。简单对象访问协议（SOAP）是一个简化的协议，通常用于基于 XML 的分散分布式网络环境中的信息交换。网络服务描述语言（WSDL）是用于描述和查找基于 XML 的网络服务的一种规范，它为服务开发商描述其系统所需要的基本格式提供了一个简单的途径，因为开发商不必考虑底层的协议（例如 SOAP 或 XML）或编码方式（如多用途 Internet 消息扩展（MIME））。Matlab 还支持 WSDL 和 SOAP，但仅局限于*远程程序调用（RPC）网*络服务。

Matlab 可以作为一个网络服务的客户端，向服务器发出请求，并处理响应。此时，Matlab 将使用 HTTP 初始化一个服务器连接。当服务器接收到请求（包括需要执行的操作和必需的参数）后，便会向请求方返回一个响应。在 Xmethods 网站（http://www.xmethods.net）上有许多服务可供用户参考，另外，Matlab 的帮助文档也对上述的诸多协议、Matlab 网络服务支持、以及相应的示例进行了详细描述。

34.9 小结

本章主要介绍了 Matlab 的编程接口，该接口使 Matlab 的功能大大扩展。本章主要介绍了以下内容：

（1）MEX 文件可以在不进行向量化的前提下提高循环执行的速度，另外，用户只要在该文件中添加少量的接口程序，就可以充分利用已经写好的 C 函数或 FORTRAN 子程序。编译好的 MEX 文件可以像普通的 M 文件函数在 Matlab 中进行调用。

（2）在用户的程序中，Matlab 可以作为一个后台计算引擎使用，这样可以充分利用 Matlab 的计算能力、计算效率和可视化优势。用户可以在自己的程序中读写 Matlab 的 MAT 文件，并使用简单的 load 和 save 命令将变量和数据导入或导出 Matlab。

（3）在 Matlab 中可以访问共享库。

（4）在 Matlab 中可以对串口进行直接控制，并实现与串口设备进行通信。

（5）在 Matlab 中可以直接访问源代码控制系统（SCCS）。

（6）使用标准协议，可以通过远程程序调用访问基于 XML 的网络服务。

以上这些在 Matlab 环境中添加的有力工具，可以使用户有效地解决难题或对已有的程序功能进行扩展。有关 Matlab API 的更多信息，包括 MX 函数的完整列表、MAT 文件的内部结构以及更多的示例代码，读者可以参考相应的 Matlab 帮助文档。

Chapter 35

Matlab 的 Java 扩展

35.1 JAVA 概述

Matlab 7 对 Java 编程语言进行了扩展使用。用户在每次安装 Matlab 时都需要使用一个 Java 虚拟机（JVM），该虚拟机由操作系统提供或集成到 Matlab 中。Java 解释器已被广泛应用在 Matlab 中，并构成了 Matlab 用户界面的基础。由于 Java 已被完全集成在了 Matlab 环境中，所以 Matlab 用户可以在 Matlab 中直接使用 Java 虚拟机。Java 的类、对象和方法都可以在 Matlab 的命令行和 Matlab 函数中使用。Java 集成为扩展 Matlab 功能提供了诸多途径。

本章将着重讨论 Java 和 Matlab 的集成。由于本书的目的不是讲解 Java，因此，用户在学习本章时最好能自备一本关于 Java 编程的参考书，或提前对 Java 编程语言进行学习。本章将只介绍那些在 Matlab 编程时有用的 Java 特性。

Java

Java 是一种编程语言，专门用于不同操作系统、不同类型的计算机的分布环境中。Java 程序都被编译成与操作系统平台无关的 Java 字节代码，因此，任何一台安装有 Java 虚拟机（JVM）的计算机都可以运行该代码。Java 虚拟机其实是一段程序，它能将字节代码翻译成可以在实际的计算机硬件上运行的机器代码。由于机器代码在执行时可以识别并调整不同计算机操作系统之间的差别，如：指令长度、数据存储方式等，因此，有了 Java 虚拟机，就消除了 Java 程序版本对操作系统的依赖性。同一个 Java 程序可以在任何一台安装有 JVM 的计算机上生成相同的结果。

Java 是一种面向对象的编程语言。如果用户曾经使用过 Java、C++或其他面向对象的编程语言，或者对本书前面讲到的 Matlab 类和对象很熟悉，那么在理解下面介绍的 Java 概念和术语时就会比较容易。

下面是本书在介绍 Java 语言时用到的主要概念和术语：

（1）类：一个 Java 类由数据规范（相当于变量）和操作集（相当于方法和函数）构成，所有基于该类的对象都可以使用这些规范和操作集。Java 类是针对特定种类的对象的模板定义。

（2）对象：Java 对象是某一 Java 类或子类的特定实例，就像 Matlab 结构体对象是 Matlab 的 struct 类的一个实例一样。对象中不再包含变量，而是实际的值。与该对象的类相关联的方法都可以对同类的实例对象进行操作。

（3）方法：方法是编写好的程序或操作，它是类的一部分，并且可以被该类的任何一个对象使用。Java 方法和 Matlab 类方法很相似，它们都是对特定类的对象运行操作的函数。

（4）变量或字段：变量和字段是代表某类值的名称。Java 变量是在 Java 类中定义的。字段和变量这两个术语经常可以互换使用。

（5）类库、工具包、软件包：软件包是相互关联的 Java 类的集合，例如抽象窗口工具包（Abstract Windowing Toolkit（java.awt.*））可以提供窗口和 GUI 服务；网络类库（java.net.*）可以提供互联网和通信服务。

（6）私有/公有：私有字段只在类内部有效，并且只能由公有方法或该类定义的私有方法修改。私有方法只能在类内部调用，不能在类外部调用。公有字段和公有方法则既在类内部有效，而且在类外部有效。

（7）静态/非静态：静态字段和静态方法是与类有关的概念，而不是与对象有关的概念。静态字段具有只读属性，其所有内容都无法进行修改。静态方法只能对类进行操作，无法对对象进行操作。非静态字段和方法则是与对象相关的概念，它们都可以随时进行修改。

（8）末级：末级类（Final Class）是没有子类的类。末级类的变量或字段都不能进行修改。末级类的方法也不能被其他级别的子类重载。私有方法也是一种很有效的末级方法（Final Method）。声明为公共静态末级（Public Static Final）的变量是一类只读变量，它们可以在类外部使用。习惯上，公共静态末级变量的名称全部使用大写字母。

在 Matlab 环境中可以访问任意的 Java 类，这使得 Matlab 的功能得到了极大的扩展。在本章的后续部分，我们将介绍几种 Java 扩展的实例。

为什么要使用 Java

为什么每个人都想使用 Java 呢？原因是多方面的。任何熟悉面向对象编程语言的用户，或能够使用 Matlab 的 MEX 文件编程的用户，都可以很轻松地使用 Java。用户可以通过访问现有的 Java 类和方法使自己的 Matlab 程序功能更加强大。另外，用户也可以使用预先编写的特定 Java 类在自己的程序中添加新的功能。最后，用户可以创建自己的 Java 类，并在 Matlab 中访问它们。目前，Matlab 可以自动实现 Java 和 Matlab 数据类型之间的转换，能够方便地在 Matlab 数据类型和 Java 对象之间传递数据。Matlab 这种对 Java 数组的内在支持，使得用户能够轻而易举地实现 Matlab 和 Java 之间的交互。

35.2 Java 的类

Java 的类是 Java 的基础。在 Matlab 中，有些标准的类和专为 Matlab 设计的类可以被 Matlab 自动识别并使用。但是，如果用户希望让 Matlab 识别并使用其他的附加类（包括本节讲的 Java 类），就需要执行两个步骤：首先，必须在 classpath.txt 文件中添加附加类所在的完整路径，以告知 Matlab 附加类.class 或.jar 文件所在的位置；然后，必须使用整个类和

软件包的名称来引用这些类，或者使用 import 命令将类或软件包导入 Matlab 工作区中。

在 Matlab 中使用 Java 类

位于$toolbox/local 目录下的全系统文件 classpath.txt 指定了定义 Java 类的位置。用户可以通过编辑此文本文件，使所有的 Matlab 用户都可以使用新添加的类。如果将此文件拷贝到$home/matlab 目录下或用户当前的工作目录下，那么对此文件的修改将只影响当前用户。为了能够使用某个.class 文件，需要将该.class 文件所在的目录添加到 classpath.txt 文件中。如果将包含各种类软件包的顶层目录添加到 classpath.txt 文件中，就可以使整个类软件包可用。将.jar 文件的完整路径（包括文件名）添加到 classpath.txt 文件中，就可以使整个压缩 Java 文档可用。需要注意的是，classpath.txt 文件只在 Matlab 启动时才被读取，因此，如果对该文件进行了修改，只有当 Matlab 重新启动时，修改才会生效。为此，Matlab 7 提供了 javaclasspath、javaaddpath、javarmpath 三个函数使 Matlab 能在执行周期内动态识别 Java 类路径的修改。

指定了 Java 类的位置后，用户就可以通过使用完整的类名来调用该类。例如：java.lang.String 将调用 java.lang 类库中的 String 类。

35.3 Java 的对象

Java 对象既可以使用 Java 语法创建，也可以使用 Matlab 语法来创建。例如，下面的代码：

```
>> myFrameA = java.awt.Frame('A Cool Window');              % Java syntax
>> myFrameB = javaObject('java.awt.Frame','Another Window');% MATLAB syntax
```

分别用 Java 语法和 Matlab 语法创建了两个对象，它们都是 java.awt.Frame 类的实例。在第二条语句中，javaObject 函数通常用于函数内部的特殊条件中，因此很少使用。为了提高用户程序的通用性，建议使用 Java 语法。

import 函数是在 Matlab 中引用一个给定类的便捷函数。例如，下面的代码：

```
>> import java.awt.Frame
>> myFrameA = Frame; myFrameB = Frame;
```

将创建两个 java.awt.Frame 类对象。使用 import 函数后，直接引用 Frame 就表明引用 java.awt.Frame 类。另外，import 函数还可以将整个类库导入 Matlab。例如，下面的一行代码：

```
>> import java.awt.* java.net.* com.mathworks.ide.help.HelpBrowser
```

将所有的 java.awt 类和 java.net 类，以及整个 com.mathworks.ide.help.HelpBrowser 类导入 Matlab。将整个类库导入后，用户就可以使用其中的某一个类。例如，下面的代码：

```
>> import java.awt.*
>> myButton = Button('Stop');
```

导入了整个 java.awt 类库，并创建了一个 java.awt.Button 类的实例。单独执行不带参数的

import 命令将返回当前的导入列表（即已经导入的类的列表）。使用 clear import 命令将会从当前的工作区中清空导入列表。如果用户在函数体内部导入类，则当函数返回时，导入列表也会被清空。

Matlab 中 Java 对象的引用

有一点需要读者注意：在 Matlab 中，Java 对象是被引用的，而不是通过赋值或值的传递进行拷贝的。无论 Java 对象是否被赋值都会创建一个新的引用。当 Java 对象作为函数的输入参数传递给函数时，该参数名也只是原始 Java 对象的一个引用而已。上述叙述看起来也许不是很好懂，下面我们通过一个例子加以说明。首先请看下面给出的这个测试函数：

```
function javatest(obj)
% Test java object references

disp(obj.getlabel)                        % Display the object label
newRef=obj;                               % Create a new reference
set(newRef,'Label','Label One')           % Change the label using set()
disp(newRef.getLabel)                     % Display the new label
setLabel(newRef,'Label Two')              % Use the setLabel method
disp(newRef.getLabel)                     % Display the label again
newRef.setLabel('Label Three')            % Use Java object method syntax
disp(newRef.getLabel)                     % Display the label
```

为了演示上面的 javatest 函数的用法，我们需要创建一个 Button 类的对象，代码如下：

```
>> myBut = java.awt.Button('Label Zero')
myBut =
java.awt.Button[button1,0,0,0×0,invalid,label=Label Zero]
```

之后，我们就可以将 Button 传递给 javatest 函数了：

```
>> javatest(myBut)
Label Zero
Label One
Label Two
Label Three
```

这样，Matlab 基本工作区中的 Button 对象就被修改了！我们可以通过下面的代码查看 myBut 对象的变化：

```
>> myBut
myBut =
java.awt.Button[button1,0,0,0×0,invalid,label=Label Three]
```

另外，由于 newRef 是函数内部声明的对象引用，因此，当函数结束时，该引用应被清除，Matlab 的基本工作区中应该不存在该引用，下面的代码证明了这一点：

```
>> newRef
??? Undefined function or variable 'newRef'.
```

上面的结果再次说明，Java 对象是通过引用来传递的，而不是通过具体的值来传递的，

在对象之间互相赋值时，则会创建一个新的引用。由于函数内的 newRef 和基本工作区中的 myBut 所指的是同一个对象，因此，对 newRef 引用所作的修改将会影响 myBut 对象。在以前的 Matlab 中，Java 对象不是通过引用传递的，而是通过具体的值传递和赋值的，这一点请习惯使用老版本的用户要特别注意。

Java 对象由对象构造器中的方法创建，并由 clear 函数清除。因为每次赋值都将创建一个新的对象引用，所以 Java 对象无法进行复制，即无法得到任何 Java 对象的拷贝。

Java 数组虽然可以被复制，但是新数组的元素仍是由原始对象创建的一个新的引用。

35.4 Java 的方法

Java 类由数据定义和操作集构成。这些数据和操作集只能作用于该类的对象，或者由该类创建的某个对象执行。Java 中的操作集被称为方法，它和 Matlab 中的操作符和函数是等效的。

在 Java 对象上调用方法

在前节的例子中，javatest 函数使用了 3 种不同的方法来修改 Button 的 Label 属性值，即：

```
set(newRef,'Label','Label One')       % Change the label using set()
setLabel(newRef,'Label Two')          % Use the setLabel method
newRef.setLabel('Label Three')        % Use Java object method syntax
```

其中，第一种方法使用重载的 set 函数对 Java 对象进行赋值，set 函数的语法为：set(object,property,value)；第二种方法使用 java.awt.Button 类的 setLabel 方法进行赋值，其语法为：method(object,value)；第三种方法使用 Java 语法进行赋值：object.method(value)。以上 3 种赋值方法本节都将用到。

除上述 3 种方法外，还有一种方法可以在一个 Java 对象上调用某一方法，即使用 javaMethod 命令来调用方法，下面是一个简单示例：

```
javaMethod('setLabel',newRef,'Label Four')
```

该语句也可以调用 Button 对象 newRef 上的 setLabel 方法。虽然 javaMethod 函数可以用于非常特殊的情况，但普通用户一般很少使用，所以，通常建议用户使用 Java 或 Matlab 语法来调用方法。

获取类和对象的信息

我们知道，类是变量和方法的集合，而对象是类或子类的实例。每个对象都继承了类的方法和变量。使用 method 命令可以查看指定类的所有可用的公有方法的列表，例如，下面的代码显示了 java.lang.Double 类的所有公有方法的列表：

```
>> methods java.lang.Double

Methods for class java.lang.Double:

Double       doubleToLongBits     intValue       parseDouble     strncmp
```

```
Number        doubleToRawLongBits  intersect        permute      strncmpi
Object        doubleValue          isInfinite       reshape      toString
byteValue     eq                   isNaN            setdiff      transpose
char          equals               isletter         setxor       tril
compare       eval                 ismember         shortValue   triu
compareTo     evalc                isspace          sort         unique
ctranspose    fieldnames           longBitsToDouble    str2double   valueOf
diag          findstr              longValue           str2num      wait
disp          floatValue           ne                  strcmp
display       getClass             notify              strcmpi
double        hashCode             notifyAll           strmatch
```

任何一个java.lang.Double类对象都可以使用上面显示的方法函数。在通常情况下，只为methods命令传递一个类名参数就可以满足要求，但如果用户需要查看每个方法函数的详细信息，就需要为method命令传递第二个参数"-full"。例如，下面的代码可以生成java.lang.Double类的更详细的方法列表（为了节省空间，我们没有把显示结果完全列出，而是使用一个省略号表示）：

```
>> methods java.lang.Double -full

Methods for class java.lang.Double:

Double(double)
Double(java.lang.String) throws java.lang.NumberFormatException
Number()
Object()
byte byteValue()
byte byteValue()
char % Inherited from opaque
static int compare(double,double)
abstract int compareTo(java.lang.Object)
int compareTo(java.lang.Object)
int compareTo(java.lang.Double)
ctranspose % Inherited from opaque
diag % Inherited from opaque
disp % Inherited from opaque
display % Inherited from opaque
double % Inherited from opaque
static long doubleToLongBits(double)
static long doubleToRawLongBits(double)
abstract double doubleValue()
double doubleValue()
eq % Inherited from opaque
boolean equals(java.lang.Object)
boolean equals(java.lang.Object) % Inherited from java.lang.Object
(...)
```

除method命令外，Matlab 7还提供了methodview函数用于执行相同的操作，只不过该函数将会创建一个新的窗口用于显示类的方法列表。如果在一个对象实例上调用了该类中没有的方法，Matlab会在其内置函数中查找是否存在相同名称的函数，如果有，便调用

该函数。如果没有同名的内置函数，则会给出一个消息，指出方法没有找到。

除了方法函数外，有时用户还需要查看类或对象中的公有字段或变量的列表，这时，可以使用 fieldnames 命令（详细用法见第 8 章）。在下面的代码中，fieldnames 命令将被重载，用于返回 Java 类或对象的公有字段或变量的信息：

```
>> dObj = java.lang.Double(5.0);

>> fieldnames(dObj)
ans =
    'POSITIVE_INFINITY'
    'NEGATIVE_INFINITY'
    'NaN'
    'MAX_VALUE'
    'MIN_VALUE'
    'TYPE'

>> fieldnames(dObj,'-full')
ans =
    'static final double POSITIVE_INFINITY'
    'static final double NEGATIVE_INFINITY'
    'static final double NaN'
    'static final double MAX_VALUE'
    'static final double MIN_VALUE'
    'static final java.lang.Class TYPE'

>> fieldnames(java.awt.Dimension)
ans =
    'width'
    'height'

>> fieldnames(java.awt.Point,'-full')
ans =
    'int x'
    'int y'
```

用户可以使用 object.field 语法访问一个 Java 对象的公有数据字段，如下例所示：

```
>> dObj.MAX_VALUE
ans =
   1.7977e+308
```

35.5 对象属性

和句柄图形对象一样，Matlab 中的 Java 对象也拥有属性。Matlab 同样使用 get 和 set 方法来访问标准的 Java 类方法，以及向 Java 对象添加通用属性。例如，java.awt.Frame 对象有一个'Background'属性，Matlab 的 get 和 set 方法可以通过执行 Java 的 getBackground 和 setBackground 方法，来修改这一属性。

为了演示 get 和 set 方法的用法，我们再看一个例子。首先，我们创建一个 java.lang.Double 对象：

```
>> dObj = java.lang.Double(6.0)
dObj =
6.0.
```

我们可以使用 get 方法检测 dObj 对象的属性：

```
>> get(dObj)
        Class = [ (1 by 1) java.lang.Class array]
        Infinite = off
        NaN = off

        BeingDeleted = off
        ButtonDownFcn =
        Children = []
        Clipping = on
        CreateFcn =
        DeleteFcn =
        BusyAction = queue
        HandleVisibility = on
        HitTest = on
        Interruptible = on
        Parent = []
        Selected = off
        SelectionHighlight = on
        Tag =
        Type = java.lang.Double
        UIContextMenu = []
        UserData = []
        Visible = on
```

Get 方法可以获得 Matlab 对象的通用属性，包括'Tag'和'UserData'属性。另外还有 3 个与特定对象相关的属性：'Infinite'、'Class'、'NaN'，它们全都是只读属性。使用 get 和 set 命令可以访问和修改对象的属性，如下例所示：

```
>> get(dObj,'Class')
ans=
class java.lang.Double
>> set(dObj,'Tag','MyDouble')
>> get(dObj,'Tag')
ans =
MyDouble
```

> 与句柄图形对象不同，Java 对象没有句柄。Java 对象不是 root 对象的子类，并且不能使用 findobj 函数或其他 Matlab 搜索技术进行查找。

35.6　数据交换

Java 的类和对象与 Matlab 的类和变量是不同的，因此在 Java 和 Matlab 的不同对象或变量之间移动数据需要进行数据格式的转换。

数据类型自动转换

Matlab 和 Java 与其他所有有类型编程语言和编程环境一样，都需要使用强制转换（coercion）或条件转换（casting）（大多数程序员都熟悉这两个概念），在不同的数据类型或类之间传递数据。每个类都至少包含一个构造器方法，用于从不同的源数据类型中创建对象。例如，针对 Matlab 的 integer 数据类型的数学运算在执行时，首先将整数值转换为双精度值，然后执行运算，最后再将计算结果转换为原来的 integer 数据类型。不同的整数数据类型不能混合在同一个表达式中，但可以通过显式转换再进行计算，如下例所示：

```
>> x = uint8(95)
x =
    95

>> y = uint16(43)
y =
    43

>> z = x+y
??? Error using ==> plus
Integers can only be combined with integers of the same class, or scalar
doubles.

>> z = y + uint16(x)
z =
    138

>> class(z)
ans =
uint16
```

Matlab 可以自动将 Matlab 本身的数据类型转换为所需的标准 Java 类，这样就掩饰了进行 Java 数据类型转换的复杂性。在通常情况下，Matlab 数值数据类型将被转换成 Java 目标数据类型。Matlab 字符串和字符数组将被转换成 java.lang.String 类。Java 对象的单元数组将被转换成对象的 Java 数组。注意：Matlab 数组和 Java 数组是不同的。

在通常情况下，从 Java 方法返回到 Matlab 的 Java 对象不会自动将其数据类型转换为 Matlab 的数据类型，这样做是为了使 Java 方法可以继续使用这些数据。但也有特例。从 Java 方法返回的数值一般会被转换成 Matlab 数据类型：其中标量会被转换成 Matlab 的 double 类型，而数值数组会被转换成最节省存储空间的 Matlab 数组。返回到 Matlab 的 java.lang.String 对象会被转换成字符数组。Java 的 String 数组可以使用 char 函数转换成字符串单元数组。在 Matlab 中创建的 Java 对象将保持它们原始的对象格式不变，包括 java.lang.Double 和 java.lang.String 对象。

显式数据类型转换

除了自动数据转换外，用户也可以在程序中进行显式数据转换。任何属于 Java 的 Numeric 类、Numeric 子类或包含 toDouble 方法的 Java 类的对象，都可以转换成 Matlab 的 double 类型。任何属于 String 类或包含 toString 方法的 Java 类的 Java 对象，都可以转换成 Matlab 的 char 类型。读者可以通过下面的例子验证上述结论：

```
>> wObj = java.lang.Double(3.12)
wObj=
3.12

>> wObj.intValue
ans=
       3

>> class(ans)
ans =
double

>> class(double(wObj))
ans =
double

>> sObj = java.lang.String('A Java String')
sObj =
A Java String

>> class(sObj)
ans =
java.lang.String

>> sObj. toUpperCase
ans=
A JAVA STRING

>> class(ans)
ans =
java.lang.String

>> butObj = java.awt.Button('OK')
but =
java.awt.Button[button1,0,0,0×0,invalid,label=OK]

>> butObj.getSize
ans =
java.awt.Dimension[width=0,height=0]

>> class(ans)
ans =
java.awt.Dimension
```

从上面的代码可以发现，Java 对象具有的一些特性通常可以使用字段名来访问，这有点类似于Matlab处理结构体的方式。struct函数可以用于将Java对象转换为Matlab结构体，同样，cell函数可以用于将Java对象或数组转换为Matlab的单元数组。

显示Java对象

当用户使用disp函数或在命令行忽略分号来显示Java对象列表时，实际上Matlab内部将使用toString方法来显示该对象。对于大多数Java对象而言，toString方法将返回一个字符串，其中包含了类名和与该对象相关的变量信息。下面给出了两个显示Java对象的例子：

```
>> butObj = java.awt.Button('OK')
butObj =
java.awt.Button[button0,0,0,0×0,invalid,label=OK]

>> f = java.awt.Frame('My Frame')
f =
java.awt.Frame[frame0,0,0,0×0,invalid,hidden,layout=java.awt.Border
          title=My Frame,resizable,normal] % (line wrapped for printing)
```

Numeric 和 String 类及其子类的 Java 对象的显示方式与上述方式不同。Numeric Java 类型的数值将被作为字符串显示；而 String 类型会被作为字符数组显示。

在某些方面，Java 对象和 Matlab 结构体非常相似。Java 类的字段（与 fieldnames 命令所列出的字段相同）可以被转换成 Matlab 结构体，并可以使用标准的 Matlab 语法访问，如下所示：

```
>> b = struct(butObj)
b =
        TOP_ALIGNMENT : 0
     CENTER_ALIGNMENT : 0.5000
     BOTTOM_ALIGNMENT : 1
       LEFT_ALIGNMENT : 0
      RIGHT_ALIGNMENT : 1
                WIDTH : 1
               HEIGHT : 2
           PROPERTIES : 4

             SOMEBITS : 8
            FRAMEBITS : 16
              ALLBITS : 32
                ERROR : 64
                ABORT : 128

>> b.PROPERTIES
ans =
    4

>> class(ans)
ans =
double
```

上述转换特性为获取 Java 类的所有公有字段或变量的名称和数值提供一种有效的方法。但如果要修改这些值，就必须要使用对象中的方法。注意：上述特殊的字段大都可能是公有静态末级类（这一类的字段名全为大写字母），因此，很多都无法进行修改。

35.7 Java 数组

我们在前几节讨论了如何在 Java 方法和 Matlab 环境之间传递单个 Java 对象。但有些方法不仅能够处理单个对象，还可以处理一组对象，有时甚至必须以一组对象作为输入参数，有些方法则可以返回一组对象。因此，我们需要用到 Java 数组的概念。Java 数组即指

一组 Java 对象，它既可以通过使用 Java 方法产生，也可以通过使用 Matlab 命令创建。

Java 数组是基于 C 语言模型的，它在结构以及元素访问方式上都与 Matlab 数组截然不同。不过，在大多数情况下，用户不必担心这种区别会带来麻烦，因为 Matlab 可以内在地进行这两类数组之间的转换。

Java 数组结构

Java 数组总是一维的；二维的 Java 数组通常被表示成以 Java 数组为元素的数组；三维的 Java 数组则表示成以二维 Java 数组为元素的数组；依此类推，*n* 维 Java 数组表示成以 *n*-1 维 Java 数组为元素的数组。Java 数组的这种结构有点类似于 Matlab 中的嵌套单元数组。与单元数组类似，嵌套 Java 数组也可以有不同的长度。例如：二维 Matlab 字符数组必须是矩形的，即每一行都必须有相同的列。然而，字符串单元数组的各个单元就可以有不同的长度。Java 数组中同样也可以包含不同长度的数组。这种可以包含不同长度单元的数组有时被称为不规则数组（ragged arrays）。当 Java 方法返回的数组是不规则数组时，Matlab 在转换时会将它们存储在单元数组中。

访问 Java 数组中的元素

和 C 语言一样，Java 数组的标号也是从 0 开始的，而 Matlab 的数组标号则是从 1 开始的。因此，Java 程序需要使用从 0 到 *N*-1 的标号来访问长度为 *N* 的数组中的元素；而 Matlab 则需要使用从 1 到 *N* 的标号来访问长度为 *N* 的数组中的元素。例如，在 Matlab 中，使用 myArray(4)表示访问一个 Java 数组的第四个元素；而在 Java 程序中，需要使用 myArray[3] 来访问该元素。又比如，在 Java 程序中使用 myArray[2][4]访问的 Java 数组元素如果在 Matlab 中访问，则需要使用 myArray(3,5)。不过，用户不必担心 Java 和 Matlab 的这一差别，因为 Matlab 将这一区别隐藏在了内部处理中，用于不必显式进行处理，因此，用户可以放心地在 Matlab 环境中使用正常的 Matlab 语法来访问 Java 数组或执行 Java 方法。下面的几个例子向读者演示了当在 Matlab 中访问 Java 数组或执行 Java 方法时，Matlab 会对标号进行透明转换：

```
>> but1Obj = java.awt.Button('STOP')
but1Obj =
java.awt.Button[button1,0,0,0×0,invalid,label=STOP]

>> but2Obj = java.awt. Button('GO')
but2Obj =
java.awt.Button[button2,0,0,0×0,invalid,label=GO]

>> butArray = [but1Obj,but2Obj]
butArray =
java.awt.Button[]:
    [java.awt.Button]
     [java.awt.Button]

>> class(butArray)
ans =
java.awt.Button[]

>> size(butArray)
```

```
ans =
    2       1

>> getLabel(butArray(1))    % Functional syntax for the getLabel method
ans =
STOP

>> butArray(2).getLabel     % Java syntax for the getLabel method
ans =
GO

>> butArray(2).setLabel('YIELD')

>> but2Obj
but2Obj =
java.awt.Button[button2,0,0,0×0,invalid,label=YIELD]
```

注意：butArray 是 but1Obj 对象和 but2Obj 对象的引用。对引用所作的修改将会影响原来的对象。

创建 Java 数组

要创建一个 Java 数组，可以使用 Java 方法通过下列操作完成：①将对象或数组串联起来。②从已有的 Java 数组中提取部分元素。③对数组元素赋值。④使用 javaArray 函数。

将 Java 对象或数组串联起来创建 Java 数组需要用到标准的数组操作符[ ]和 cat 函数。如果所串联的对象属于同一类，那么串联后的数组也将是同类数组。如果所串联的对象属于同一类的数组，那么将采用层叠方式创建 Java 数组。以上两种方式所得到的数组长度将是各个独立数组长度的和。如果所串联的对象是二维数组，并且第二维的长度不同，那么所得到的数组将是一个不规则数组。如果所串联的对象属于不同的类，那么所得到的数组将是 java.lang.Object 类的数组，其长度等于所串联的数组的数量。

对于普通的 Matlab 数组，用户可以将一个 Java 对象赋给该数组的某一元素，来创建 Java 数组，如下例所示：

```
>> bigButArray(3,4) = java.awt.Button('YES')
bigButArray =
java.awt.Button[][]:
    []      []      []                          []
    []      []      []                          []
    []      []      []          [java.awt.Button]
```

有时候，扩展一个数组很可能会导致不可预料的结果。由于 Java 数组可以是不规则的数组，因此，通过赋值来扩展已有的 Java 数组不一定总会生成矩形数组，如下例所示（该例使用了前例中的 bigButArray）：

```
>> bigButArray(4,5) = java.awt.Button('No')
bigButArray =
java.awt.Button[][]:
    [4 element array]
     [4 element array]
     [4 element array]
     [5 element array]
```

上例在已有的Java数组bigButArray中添加了第4个元素，该元素是含有Button对象的一个长度为5的Java数组。从上面的结果可以看出，bigButArray中原来的3个元素的长度保持不变（仍为4）。因此，最终得到的数组不是矩形的，也就是说是不规则的。对于这类数组，Matlab将以单元数组的方式存储。

我们知道，当将空矩阵赋给Matlab矩阵的整行或整列时，Matlab就会删除该行或该列，同时矩阵的维数也会相应减小。但由于Java矩阵有可能是不规则的，所以我们不能以相同的方式处理Java矩阵。当将空矩阵赋给Java矩阵的整行或整列时，该行或该列的元素将被简单地赋以Java的NULL字符（Matlab在转换时将把它转换为空矩阵），而Java矩阵的维数不会发生变化。

javaArray函数可以用来创建一个指定类的未被填充的Java数组（即数组的每个元素都是NULL值），其语法为：javaArray('java_class',m,n,p,...)，其中java_class是一个完全限定的Java类名（包括软件包或类的名称），m、n、p是数组的维数。例如，要创建一个包含Button对象的3×4的空Java数组，可以使用下面的代码：

```
>> buttonArray = javaArray('java.awt.Button',3,4)
buttonArray =
java.awt.Button[][]:
     []        []        []        []
     []        []        []        []
     []        []        []        []
```

然后，我们就可以使用标准的Matlab赋值语法和Button构造器方法将一个新的Button对象填充到数组中，如下面的代码所示：

```
>> buttonArray(2,2) = java.awt.Button('MAYBE')
buttonArray =
java.awt.Button[][]:
     []                       []        []        []
     []    [java.awt.Button]            []        []
     []                       []        []        []
```

访问Java数组元素

Java对象数组的元素可以使用普通的Matlab数组寻址语法访问，如下例所示：

```
>> dArray = javaArray('java.lang.Double',3,4)
dArray =
java.lang.Double[][]:
      []      []      []      []
      []      []      []      []
      []      []      []      []

>> for i = 1:3
      for j = 1:4
         dArray(i,j) = java.lang.Double(i*10+j);
      end
   end

>> dArray
```

```
dArray =
java.lang.Double[][]:
     [11]     [12]     [13]     [14]
     [21]     [22]     [23]     [24]
     [31]     [32]     [33]     [34]
```

注意：上述结果是一个按行排列的 3 元素 Java 数组，其中每个元素由 4 个按列排列的元素组成。下面的结果表明，在 Java 中使用 dArray[0][2]所引用的元素和在 Matlab 中使用 dArray(1,3)所引用的元素是同一元素：

```
>> dArray(1,3)
ans =
13.0
```

我们也可以使用下面的代码查看 dArray 的第二个元素，它应该是一个有 4 个元素的 Java 数组：

```
>> dArray(2)
ans =
java.lang.Double[]:
    [21]
    [22]
    [23]
    [24]
```

说明：上述结果仅对 Java 数组成立，对 Matlab 数组不适用。例如，给定一个双精度类型的 3×4 的 Matlab 数组 D，那么 D(4)和 D(1,2)指的将是同一数组元素。

子数组

用户可以使用与前面相似的 Matlab 语法来访问 Java 数组的子数组。在下面的例子中，我们将采用前面的 Java 数组 dArray 和与之相对应的 Matlab 数组 D 来演示对子数组的访问。首先，dArray 和 D 给定如下：

```
>> dArray
dArray =
java.lang.Double[][]:
     [11]     [12]     [13]     [14]
     [21]     [22]     [23]     [24]
     [31]     [32]     [33]     [34]
>> D = double(dArray)
D =
     11     12     13     14
     21     22     23     24
     31     32     33     34
```

然后，我们可以通过标准的 Matlab 语法创建基于这两个数组的子数组：

```
>> dArray(2,2:3)
ans  =
java.lang.Double[]:
```

```
        [22]
        [23]

>> D(2,2:3)
ans =
        22      23
```

Matlab 使用按列的方法存储数组元素，而 Java 则使用按行的方法存储数组元素。例如，一个 2×2 的 Matlab 数组，其存储形式为[11 21 12 22]，而同样的 2×2 的 Java 数组（指一个由两个元素构成的 Java 数组，其中每个元素又包含两个元素），其存储形式为[11 12][21 22]。基于上述差别，在使用冒号运算符（:）由一个数组创建一个子数组时，对于 Matlab 和 Java 所得到的结果是不相同的，如下例所示：

```
>> dArray(:)
ans=
java.lang.Double[]:
        [11]
        [12]
        [13]
        [14]
        [21]
        [22]
        [23]
        [24]
        [31]
        [32]
        [33]
        [34]
>> D(:)
ans =
      11
      21
      31
      12
      22
      32
      13
      23
      33
      14
      24
      34
```

冒号运算符将数组视为一个单独的向量，使用一条语句为 Java 数组或子数组的每个元素赋值。请看下面给出的例子：

```
>> dblArray = javaArray('java.lang.Double',3,4)
dblArray =
java.lang.Double[][]:
          []        []        []        []
          []        []        []        []
```

```
        []        []        []        []

>> dblArray(:) = java.lang.Double(0)
dblArray =
java.lang.Double[][]:
        [0]       [0]       [0]       [0]
        [0]       [0]       [0]       [0]
        [0]       [0]       [0]       [0]

>> dblArray(2,:) = java.lang.Double(1)
dblArray =
java.lang.Double[][]:
        [0]       [0]       [0]       [0]
        [1]       [1]       [1]       [1]
        [0]       [0]       [0]       [0]

>> dblArray(:,3) = java.lang.Double(4)
dblArray =
java.lang.Double[][];
        [0]       [0]       [4]       [0]
        [1]       [1]       [4]       [1]
        [0]       [0]       [4]       [0]
```

注意：上述所有元素都是对同一对象的多次引用。其中的原始构造器赋值语句 dblArray(:)=java.lang.Double(0)用于创建一个包含 0 值的 java.lang.Double 对象，并通过对该对象的 12 次引用来填充数组 dblArray。在此语句之后的两个赋值语句都创建了一个新的对象，并对该对象进行了多次引用。

复制数组

对 Java 数组进行赋值同样也会创建新的引用，下面的代码：

```
>> xArray = dArray
xArray =
java.lang.Double[][]:
    [11]        [12]        [13]        [14]
    [21]        [22]        [23]        [24]
    [31]        [32]        [33]        [34]
```

将创建一个 dArray 的引用，并命名为 xArray。另外，我们还可以创建数组的单个元素的引用，如下面的代码所示：

```
>> xObj = dArray(2,3)
xObj =
23.0
```

上例中，xObj 和 dArray(2,3)引用的是同一个对象。

如果用户将数组（该数组包含两个或多个层次）的最顶层元素赋值给一个变量，那么该变量将是对顶层数组的引用，如下例所示：

```
>> yArray = dArray(2)
yArray =
```

```
java.lang.Double[]:
     [21]
     [22]
     [23]
     [24]

>> yArray(2) = java.lang.Double(0)
yArray =
java.lang.Double[]:
     [21]
     [ 0]
     [23]
     [24]

>> dArray
dArray =
java.lang.Double[][]:
     [11]     [12]     [13]     [14]
     [21]     [ 0]     [23]     [24]
     [31]     [32]     [33]     [34]
```

上面的例子通过引用 dArray 数组的第二个顶层元素创建了 yArray。

用户还可以通过将原来 Java 数组元素的一个子集分配给一个新的变量，来创建一个新的数组，这一操作在 Java 中称为克隆（cloning）。如下例所示：

```
>> zArray = dArray(1:2,2:3)
zArray =
java.lang.Double[][]:
    [12]          [13]
    [ 0]          [23]
```

上例中，zArray 是一个新的 Double 类型的数组，它所包含的值都是对原始对象的引用。例如，zArray(1,1)和 dArray(1,2)将指向同一个对象。如果用户使用一个新的对象来替代 zArray 中的某个元素，dArray 将不受任何影响，如下例所示：

```
>> zArray(2,2) = java.lang.Double(-1)
zArray =
java.lang.Double[][]:
    [12]          [13]
    [ 0]          [-1]

>> dArray
dArray =
java.lang.Double[][]:
    [11]          [12]          [13]          [14]
    [21]          [ 0]          [23]          [24]
    [31]          [32]          [33]          [34]
```

上例中的元素替换实际上是使用一个新对象引用来代替原来的对象引用，因此这种替换不会对 dArray 产生任何影响，并且 zArray 中的其他元素也仍旧保持原来的引用。只有对这些元素的属性进行修改时，才会影响到原始的元素。

使用冒号运算符可以复制整个 Java 数组，如下面的代码所示：

```
>> xArray = dArray(:,:)
xArray =
java.lang.Double[][]:
     [11]     [12]     [13]     [14]
     [21]     [ 0]     [23]     [24]
     [31]     [32]     [33]     [34]

>> xArray(2,3) = dArray(3,4)
xArray =
java.lang.Double[][]:
     [11]    [12]    [13]    [14]
     [21]    [ 0]    [34]    [24]
     [31]    [32]    [33]    [34]

>> dArray
dArray =
java.lang.Double[][]:
     [11]    [12]    [13]    [14]
     [21]    [ 0]    [23]    [24]
     [31]    [32]    [33]    [34]
```

由上述结果我们发现，xArray(2,3)处的元素实际上是对 dArray(3,4)的一个新的引用，而 dArray 数组本身不受影响。

用户也可以通过将多个 Java 数组串联起来，创建新的数组。如下例所示：

```
>> a = dArray(1,3)
a =
13.0

>> b = dArray(3,2)
b =
32.0

>> c = [a b]
c =
java.lang.Double[]:
      [13]
      [32]

>> c(2) = dArray(3,3)
c =
java.lang.Double[]:
      [13]
      [33]

>> a
a =
13.0

>> b
b =
32.0
```

> 通常，当用户对 Java 对象或数组进行赋值时，Matlab 都会创建新的引用。当用户对 Java 数组的子集进行赋值，或者使用冒号操作符对整个数组赋值，或者将多个 Java 数组串联起来时，Matlab 都会创建新的数组。并且新的数组包含的都是对原始数组元素的引用。

例如，下列语句

```
xArray = dArray;
yArray = dArray(2);
zObj = dArray(2,3);
```

将创建新的引用，而下列语句

```
xArray = dArray(:.:);
yArray = dArray(2,:);
zArray = [dArray,dArray];
```

创建的新数组包含的是对原始数组的某些特定元素的附加引用。

Java 数组的大小

为了提供对 Java 数组的支持，Matlab 增强了一些函数的功能。但要想让所有的 Matlab 函数都提供对 Java 数组的全面支持，目前还不太可能。例如，Matlab 函数 size 和 length，只能返回 Java 数组的顶层数组的信息，如下例所示：

```
>> myArray = java_array('java.lang.Double',3,4)
myArray =
java.lang.Double[][]:
    []        []        []        []
    []        []        []        []
    []        []        []        []

>> size(myArray)
ans =
    3        1

>> length(myArray)
ans =
    3
```

要得到 Java 数组的其他维的信息，就必须指定具体的顶层数组元素，如下例所示：

```
>> size(myArray(1))
ans =
    4        1

>> length(myArray(2))
ans =
    4
```

使用 length 函数可以得到非空 Java 数组中元素的总个数，如下例所示：

```
>> length(dblArray(:))  % the dblArray created earlier
```

```
ans =
    12
>> length(myArray(:))  % the 0-by-1 (empty) myArray
ans =
    0
>> size(myArray(:))
ans =
    0       1
```

由于 myArray 是空数组，因此，size(myArray(:))将返回 0×1 的信息，而 length(myArray(:))返回的是 0。

另外，用户也可以使用关键字 end 来定位 Java 数组（表明定位到数组最后一个元素），但只能在顶层使用，如下例所示：

```
>> dblArray(2:end)
ans =
java.lang.Double[][]:
        [1]     [1]     [4]     [1]
        [0]     [0]     [4]     [0]
>> dblArray(2,2:end)
ans =
java.lang.Double[][]:
    [0 element array]
```

35.8 Java 函数

除了前一节提到的 size 函数和 length 函数外，Matlab 还创建和修改了其他一些函数用来支持 Java 的类、对象和方法。我们在前面曾经提到，import 函数可以用于向 Java 类和软件包列表中导入新的内容，并且可以简化对 Java 类的引用。clear 函数则用来删除 Java 对象，并清空导入列表。inmem 函数也经过了修改，可以使用其第三个可选输出参数来列出所有载入到 Matlab 工作区中的 Java 类的名字。exist 函数也被进行了扩展，可以用于识别和判定 Java 对象和数组。class 函数的功能也得到了扩展，可以识别 Java 对象和数组的类。isa 函数目前也可以用于测试 Java 对象或数组的类。另外，Matlab 还新增了一个新函数 isjava，用于确定一个变量是否是 Java 对象。

fieldnames 函数可以返回 Java 类或对象的公有字段的列表。methods 函数则可以列出能对某一特定 Java 类的对象进行操作的所有方法的名称。另外，which 函数的功能也得到了扩展，它可以返回包含某一方法或方法与符号的组合的所有已装载的 Java 类的列表，如下例所示：

```
>> which -all resize
java.awt.Button.resize                              % Button method
java.awt.TextArea.resize                            % TextArea method
java.awt.Frame.resize                               % Frame method
matlab7/toolbox/matlab/graph2d/@axisobj/resize.p % axisobj method
matlab7/toolbox/matlab/graph2d/@axisobj/resize.m % Shadowed axisobj method
```

save 和 load 函数可以用于存储和载入以普通二进制格式存储在 MAT 文件中的 Java 对象和数组。注意：Java 对象无法存储成 ASCII 格式的文件。在 Matlab 中，Java 的异常事件也会被自动捕获并转换成普通的 Matlab 错误文本，这些文本可以使用 lasterr 函数捕获并显示。

下表对 Matlab 中所有支持 Java 的 Matlab 函数进行了总结。其中一些函数是标准的 Matlab 函数，它们为了支持 Java 进行了重载；而另一些函数则是 Matlab 专门创建用以支持 Java 的。

函数	描述
methods	列出一个 Java 类的所有公有方法
methodsview	在一个新的视窗列出一个 Java 类的所有公有方法
usejava	判断一个 Java 特性是否被 Matlab 支持
fieldnames	列出一个 Java 类的公有字段或变量
import	列出 Java 类和软件包的导入列表，或向该表中添加内容
inmem	将所有已装载的 Java 类的名称用一个单元数组返回
which	返回已装载的 Java 类中所有与指定方法相匹配的方法列表
class	确定一个 Java 对象的类
clear	清除工作区中的 Java 对象，或清空 Java 类导入列表
double	将 Numeric 类的 Java 对象或数组转换成 Matlab 双精度类型
char	将 String 类的 Java 对象或数组转换成 Matlab 字符数组或字符串单元数组
struct	根据一个 Java 类或对象的公有字段，创建一个结构体
isjava	判断是否是 Java 对象或 Java 数组
isa	判断一个变量是否是指定类的 Java 对象
exist	判断一个变量的自然属性，如果该变量是 Java 类，则返回 8
ismethod	判断一个方法是否是 Java 对象的方法
isprop	判断一个属性是否是 Java 对象的属性
inspect	显示一个图形接口，用于列出和修改属性值
[ ]和 cat	将多个 Java 对象或数组进行串联，创建新的 Java 数组
save	将变量（包括 Java 对象）存储到一个二进制 MAT 文件中
load	从一个二进制 MAT 文件中载入变量（包括 Java 对象）
size	返回一个 Java 数组的顶层数组的大小
length	确定一个 Java 数组的某一顶层元素的长度
javaArray	创建一个新的 Java 数组，并使用 NULL 值填充该数组
javaMethod	针对一个 Java 对象或 Java 数组调用某一 Java 方法（不推荐使用）
javaObject	创建一个 Java 对象（不推荐使用）
javaaddpath	将一个完整路径添加到动态 Java 类路径列表（dynamic Java class path）中
javarmpath	从动态 Java 类路径列表中删除一个完整路径
javaclasspath	设置和获取一个动态 Java 类路径
javachk	如果 Matlab 无法使用 Java 的某一特性，或无法得到该特性，则该函数用于产生一个错误消息

35.9 示例详解

本节将通过几个详细的例子，使读者对本章所讲的概念和工具有更深的理解，并演示如何使用 Java 对 Matlab 的功能进行扩展。遵循由浅入深的原则，我们首先从几个基本的例子开始，然后再介绍稍微复杂的函数。

例 1：whoami

本例将使用 java.net 类库中的元素，返回用户计算机的主机名和 IP 地址。代码如下：

```
function whoami
% Test function to illustrate the use of the java.net package.
%
% Mastering MATLAB 7 Java Example 1

% Use try and catch to avoid Java exception messages.
try
    me=java.net.InetAddress.getLocalHost;
catch
    error('Unable to get local host address.');
end

% Find my hostname and IP address
myname=me.getHostName;
myip=me.getHostAddress;

% and print the results.
disp(sprintf('My host name is %s',char(myname)));
disp(sprintf('My IP address is %s',char(myip)));
```

在作者的计算机上，whoami 函数运行的结果如下：

```
>> whoami
My host name is www.phptr.com
My IP address is 204.179.152.74
```

例 2：random

本例将使用 java.util 软件包来生成一个随机数。代码如下：

```
function num=random(varargin)
% Test function to illustrate the use of the java.util package.
%   Generate a random number between given limits (or 0:1).
%
% Mastering MATLAB 7 Java Example 2

% Check any input arguments.
if nargin == 0
   rmin=0; rmax=1;
elseif nargin < 3
```

```
   if nargin == 1
      lim = varargin{1};
   else
      lim = [varargin{1}, varargin{2}];
   end
   if isnumeric(lim) & length(lim) == 2
      rmin = min(lim); rmax=max(lim);
   else
      error('Invalid limits.');
   end
else
   error('Too many arguments.')
end

% Construct a Random object and generate a uniformly-distributed
% random number between the desired limits.
rNum = java.util.Random;
num = rNum.nextDouble * (rmax - rmin) + rmin;
```

上述 random 函数主要执行如下操作：首先分析输入参数；然后创建一个 Random 对象；最后使用 nextDouble 方法获得一个均匀分布的伪随机数。用户也可以用 nextGaussian 方法代替 nextDouble，从而生成一个正态分布的随机数。另外，使用其他类似的 Random 函数还可以生成 integer、long、float 类型的随机值，甚至还可以生成用户指定字节数的随机数。

例 3：hithere

本例将创建一个简单的对话框，其中包括一行文本和一个按钮。点击按钮，将会退出对话框。代码如下：

```
% script hithere.m
% Example script to illustrate the use of the java.awt package.
%
% Mastering MATLAB 7 Java Example 3

% Create a Frame object and specify a 2×1 grid layout style.
dbox=java.awt.Frame('Hi There!');
dbox.setLayout(java.awt.GridLayout(2,1));

% Specify the window location, size and color. Use Java syntax
%   for the location, hybrid syntax for the size, and MATLAB
%   syntax to set the background color.
dbox.setLocation(50,50);
resize(dbox,200,100);
set(dbox,'Background',[.7,.8,.9]);

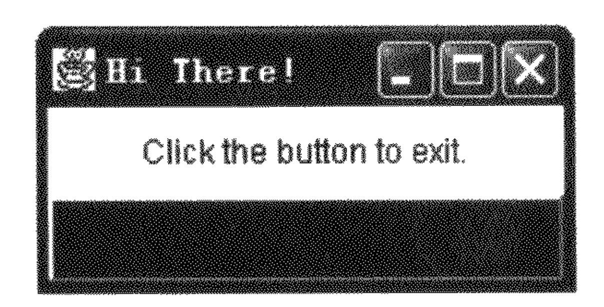
% Create a text label and a bright red button.
txt=java.awt.Label('Click the button to exit.',1);
but=java.awt.Button('Exit Button');
set(but,'Background',[1,0,0]);
```

```
% Define a callback for the button.
set(but,'MouseClickedCallback','dbox.dispose')

% Attach the label and button to the window.
dbox.add(txt);
dbox.add(but);

% The window is hidden by default. Make it visible.
dbox.show;
```

本例使用了 M 脚本文件，没有使用 M 函数文件，以避免在使用 Button 回调时出现的有效范围问题。与句柄图形对象一样，Java 回调既可以是函数句柄，也可以是传递给 eval 函数的 Matlab 工作区中的字符串。hithere.m 脚本生成的对话框如下：



例 4：netsearch

本例使用 java.net 软件包来管理互联网通信，同时使用 java.io 软件包来读取数据流。其中的 netsearch 函数用于执行互联网搜索功能，并以单元数组的形式返回搜索引擎找到的 URL（Uniform Resource Locator）。如果利用某个搜索引擎没有找到匹配的 URL，那么该程序会尝试使用其他的搜索引擎继续搜索。本例的代码如下：

```
function ulist=netsearch(varargin)
% MATLAB Java Demo function netsearch.m
%
%  Mastering MATLAB 7 Java Example 4
%    Open a connection to an Internet search engine using
%    Java networking toolkit objects and return the URLs
%    found by the search in a cell array.

% Import the java.net and java.io toolkits to save typing.
import java.net.* java.io.*

% Define the url for a search command and a target string
%     for each host to try in order.
s_host={
        'http://www.google.com/search?q=',   '<p class=g><a href='
        'http://search.yahoo.com/search?p=', 'H=0/*-href='
        };

nhosts=size(s_host,1);      % Number of search hosts defined
ulist={};                   % Cell array to contain results

% Do some error checking.
```

```
if nargin < 1
    error('Nothing to search for.');
end
if nargout > 1
    error('Too many output arguments.');
end

% Create a search string from the input arguments.
for (idx=1:nargin)
    if ~ischar(varargin{idx})
        error('Search terms must be strings');
    else
        tmp_str=varargin{idx};

        % If this string argument contains spaces, quote the string
        % and replace the spaces with plus sign characters.
        if findstr(tmp_str,' ')
            tmp_str=['%22',strrep(tmp_str,' ','+'),'%22'];
        end

        % Build up the search string.
        if idx == 1
            s_str=tmp_str;
        else
            s_str=[s_str,'+',tmp_str];
        end
    end
end

% Start with the initial search host.
uidx=1;
hostidx=1;
while isempty(ulist) & (hostidx <= nhosts)

    % Construct a complete URL using the search string.
    s_url=[s_host{hostidx,1},s_str];

    % Create a URL connection to the search engine.
    s_con=java.net.URL(s_url);

    % Open an input stream on the connection.
    s_stream=openStream(s_con);

    % Open an input stream reader to read the stream.
    s_rdr=InputStreamReader(s_stream);

    % Open a buffered reader to read one line at a time.
    shBuf=BufferedReader(s_rdr);

    % Read the lines of the page returned by the search engine
    %  and extract any target URLs until the the page ends.
    linebuf=shBuf.readLine;
    while ~isempty(linebuf)
        linebuf=char(linebuf);
```

```
            found=findstr(linebuf,s_host{hostidx,2});
            if ~isempty(found)
                tmp=strtok(linebuf(found(1)+
                            length(s_host{hostidx,2}):end),'>');
                ulist(uidx)=strrep(tmp,'"','');
                                            % Remove extra quote characters
                uidx=uidx+1;
            end
            linebuf=shBuf.readLine;
        end
        % We are done with the page so close the connection.
        shBuf.close;
        % If a target URL was not found, try the next host.
        if isempty(ulist)
            hostidx=hostidx+1;
        end
    end
    if isempty(ulist)
        ulist=[];
    elseif length(ulist) == 1
        ulist=ulist{1};
    end
```

在上面的程序中，每个主机的 s_host 数组由两个字符串构成。第一个字符串中包含了协议、主机和用于搜索 URL 的搜索命令。当搜索到的条目以正确的格式添加到此字符串中后，就可以生成用于创建 java.net.URL 对象的完整的 URL 字符串。程序在 searchHost 对象上使用了 openStream 方法，打开了本机与搜索到的主机之间的互联网连接。在此连接基础上，程序创建了一个 InputStreamReader 对象，并将该对象传递给 BufferedReader 函数，以启动输入流的行输入过程。然后，使用了 readLine 方法每次向字符串缓冲区读取一行数据。

s_host 数组的第二个字符串是一个目标字符串（即用户需要搜索的字符串）。我们知道，网页都是由一系列文本构成的，这些文本通常用来规定网页显示的格式，另外还包含一些称为标记符（tag）的代码（一般用“<”和“>”字符括起来）。在搜索引擎找到的目标 URL（也称为命中目标）之前，一般都会有一个在该网页中找到的与目标字符串相符的字符串。上述程序在查找网页时，将逐行检测网页中是否含有目标字符串。如果有，就将该页面的 URL 提取出来，并复制到输出单元数组中。整个网页被全部搜索完后，与该页的连接将关闭。如果没有找到结果，那么程序将自动连接到下一个搜索主机，继续进行搜索。注意：由于网页的内容会随着时间而变化，因此，用户在查找时也应该适时修改 s_host 中的字符串。

例 5：winfun

本例是例 3 的扩展。本例中创建的窗口将拥有多个按钮和菜单，并带有回调。当首次调用 winfun 时，程序会调用 java.awt 工具包对象构建一个初始化状态的窗口。之后，函数

退出，但创建的 Java 对象窗口仍然存在，但是对该对象的引用（即变量）不再存在于基本工作区中。如果按下一个按钮或选择一条菜单条目，就将执行相关联的回调。回调将重新调用带有惟一参数的 winfun 函数。之后，winfun 的参数将被解析，以完成回调服务功能。如果在调用 winfun 函数时没有带有任何参数，则 Java 窗口将依然存在，只不过将被重新设置成初始状态。下面是本例的代码：

```
function winfun(varargin)
% MATLAB Java Demo function winfun.m
%
%  Mastering MATLAB 7 Java Example 5
%    Create a window with buttons, menus and text objects using
%    Java windowing toolkit objects within a function. Use a
%    local function to implement callbacks. Use persistent variables
%    to maintain visibility of the Java objects between calls.
%

% Import the entire Java Abstract Window Toolkit.

import java.awt.*

% Make sure we can find the Java objects when servicing callbacks.

persistent win ta mi abt mq ma txt x y h w

if isempty(win)

  % Initial function call: create the necessary objects.
  % Start with some text for the About Box.
  txt=[' Cool Window version 1.4  ';...
        ' MM7 example function by  ';...
        'Mastering MATLAB authors  ';...
        'Hanselman and Littlefield '];

  % Reshape it for use in the textarea as well.
  tstr=reshape(txt',1,prod(size(txt)));

  % Create a window using the flow layout model (the simplest)
  %    and set the title. Set the layout using Java syntax.
  win=Frame('Cool Java Window');
  win.setLayout(FlowLayout);

  % Specify the window location, size and color. Use Java syntax
  %    for the location and size, and use MATLAB syntax to set
  %    the background color.
  x=150; y=150; w=430; h=160;
  win.setLocation(x,y);
  win.setSize(w,h);
  set(win,'Background',[.8 .8 .8]);

  % Create some menus: a 'File' menu and a 'Help' menu.
  mf=Menu('File');
  mh=Menu('Help');

  % Create a 'Quit' menu item, define a callback to close the window,
```

```
%    and attach it to the File menu.
mq=MenuItem('Quit');
set(mq,'ActionPerformedCallback','winfun(''quit'')');
mf.add(mq);

% Create a 4x21 text area with no scroll bars (3).
ta=TextArea(tstr,4,21,3);

% Don't let the user change the text.
ta.setEditable(0);

% Set the text background color.
set(ta,'Background',[1 1 .9]);

% Create a 'Show Info' menu item and attach it to the Help menu.
%   Show or hide the textarea when the menu item is selected.
mi=MenuItem('Show Info');
set(mi,'ActionPerformedCallback','winfun(''info'')');
mh.add(mi);

% Create an 'About' menu item and attach it to the Help menu.
ma=MenuItem('About');
set(ma,'ActionPerformedCallback','winfun(''about'')');
mh.add(ma);

% Create a menubar object, add the menus, and attach
%   the menubar to the window.
mb=MenuBar;
mb.add(mf); mb.add(mh);
mb.setHelpMenu(mh);      % Move the Help menu to the right side.
win.setMenuBar(mb);

% Create some buttons.
bs=Button('Shrink');
be=Button('Expand');
bl=Button('Left');
bu=Button(' Up ');
bd=Button('Down');
br=Button('Right');
bq=Button('Quit');

% Define callbacks for the buttons.
set(bs,'MouseClickedCallback','winfun(''shrink'')')
set(be,'MouseClickedCallback','winfun(''expand'')')
set(bl,'MouseClickedCallback','winfun(''left'')');
set(bu,'MouseClickedCallback','winfun(''up'')');
set(bd,'MouseClickedCallback','winfun(''down'')');
set(br,'MouseClickedCallback','winfun(''right'')');
set(bq,'MouseClickedCallback','winfun(''quit'')');

% Specify some colors for the buttons.
set(bs,'Background',[.5,1,1]);
set(be,'Background',[.5,1,1]);
set(bq,'Background',[1,.4,.4]);
```

```
    set(bl,'Background',[.9,.9,.9]);
    set(bu,'Background',[.7,.7,.7]);
    set(bd,'Background',[.7,.7,.7]);
    set(br,'Background',[.9,.9,.9]);

    % Add the first row of buttons to the window. Objects will be
    %   positioned in the order in which they are added.
    win.add(bs);
    win.add(bl);
    win.add(bu);
    win.add(bd);
    win.add(br);
    win.add(be);

    % Attach the text area to the window, but don't display it yet.
    win.add(ta);
    ta.setVisible(0);

    % Now add the Quit button.
    win.add(bq);

    % Now that the main window has been created, it is time
    %   to create a dialog box for the 'About' menu item.
    abt = Dialog(win, 'About Winfun');
    set(abt,'Background',[.95 .95 .95]);
    abt.setLayout(GridLayout(5,1));

    % Add some lines of text
    abt.add(Label(txt(1,:)));
    abt.add(Label(txt(2,:)));
    abt.add(Label(txt(3,:)));
    abt.add(Label(txt(4,:)));

    % and a button bar to close the About box.
    bok = Button('OK');
    set(bok,'ActionPerformedCallback','winfun(''ok'')');
    set(bok,'Background',[.6 .8 .8]);
    abt.add(bok);

    % Set the size of the dialog box but don't make it visible.
    abt.setSize(180,110);

    % All done. Now show the main window and exit.
    win.setVisible(1);

  elseif nargin == 0
    % Reset the hidden window to the initial state and show it.
    abt.setVisible(0);
    ta.setVisible(0); mi.setLabel('Show Info');
    x=150; y=150; w=430; h=160;
    win.setLocation(x,y);
    win.setSize(w,h);
    win.setVisible(1);

  elseif nargin == 1   % This is a callback.
```

```
    switch (varargin{1})
      case 'shrink'   % Shrink width by 4 pixels, height by 2
        w=w-4; h=h-2;
      case 'expand'   % Expand width by 4 pixels, height by 2
        w=w+4; h=h+2;
      case 'left'     % Move 4 pixels to the left
        x=x-4;
      case 'up'       % Move 2 pixels higher
        y=y-2;
      case 'down'     % Move 2 pixels lower
        y=y+2;
      case 'right'    % Move 4 pixels to the right
        x=x+4;
      case 'info'     % Toggle visibility of the text area
        if (ta.isVisible)
          ta.setVisible(0); mi.setLabel('Show Info');
        else
          ta.setVisible(1); mi.setLabel('Hide Info');
        end
      case 'about'    % Show the About Box to the right and lower
        abt.setLocation(x+w+20,y+10);
        abt.setVisible(1);
      case 'ok'       % Hide the About Box
        abt.setVisible(0);
      case 'quit'     % Hide the windows
        abt.setVisible(0);
        win.setVisible(0);
        return
      otherwise       % Bad argument
        error('Invalid callback');
    end
    win.setBounds(x,y,w,h); win.setVisible(1);

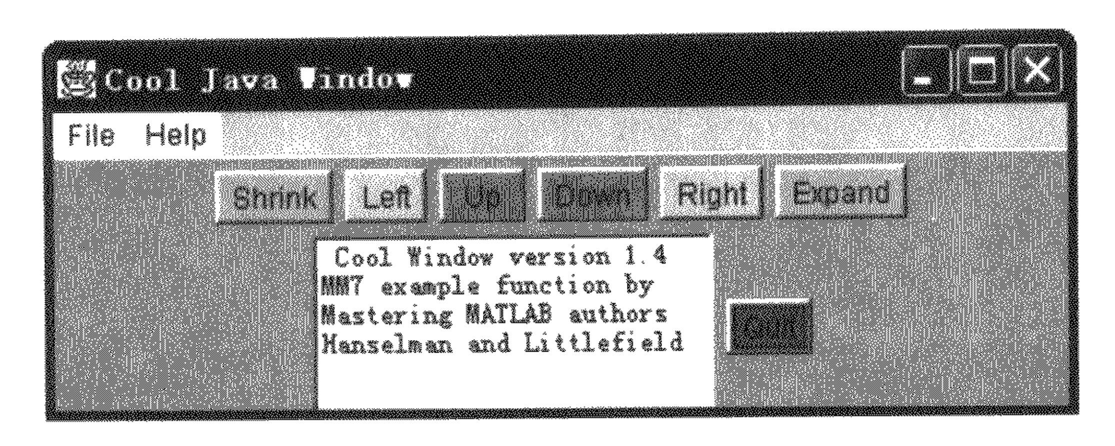
 else                 % Should not get here.
    error('Too many arguments.');
 end
```

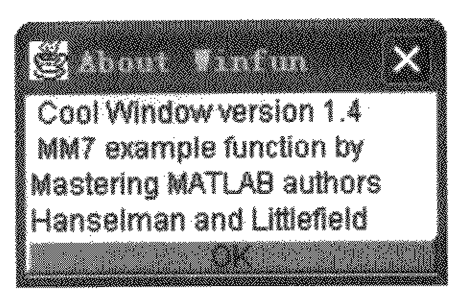
本例中，回调参数决定了 switch 语句中的哪一条 case 语句被执行。例如，当该参数为'shrink'时，函数就会使用 setBounds 方法将窗口对象 win 的水平宽度减小 2 个像素，垂直高度就减小 4 个像素。然后，使用 setVisible 方法重新绘制窗口。同样，'expand'、'up'、'down'、'left'、'right'等参数也使用相同的方法对窗口的大小或位置进行改变。

当 info 参数被调用时，其结果将取决于文本区字段对象 ta 的状态。如果 ta 的状态为可见（ta 是否可见由 isVisible 方法决定），则文本区字段被隐藏，菜单条目标签也会随着变化。如果 ta 的状态为隐藏，则使用 setVisible 方法使之可见，菜单条目标签也会随之改变。之后，程序将调用 win.setVisible 来刷新窗口。

当程序遇到 about 参数时，则会在 win 窗口的右侧显示一个 abt 对话框。此时，当程序遇到 ok 参数时，abt 对话框将被隐藏。当程序遇到 quit 参数时，win 窗口和 abt 对话框将同时消失。

下面是 winfun 函数所生成的窗口：





35.10　小结

Java 是一种面向对象的、独立于操作系统平台的编程语言。通过使用 Java 语言，可以大大扩展 Matlab 环境的使用范围。在 Matlab 7 安装时，Java 虚拟机就已经被集成到 Matlab 开发环境中供用户使用。在 Matlab 中，用户既可以通过命令行，也可以使用 Matlab 函数对 Java 的类、对象和方法进行操作。Java 集成为 Matlab 的功能扩展提供诸多途径。目前，虽然在 Matlab 中集成了 Java 功能，但用户仍不能从 Java 中调用 Matlab 函数。这一功能将有待于日后的 Matlab 版本来实现。

本章所举的的例子仅仅是 Matlab 的 Java 扩展的很小一部分内容，Matlab 的帮助文档中有更详细的 Matlab 的 Java 扩展的例子，以及与本章所讨论的问题相关的内容。但有一点请读者记住：Java 对象最重要的属性在于，它是通过引用而不是通过具体的值来传递的。

> 重要提示：Matlab 所使用的 Java 对象都是对原对象的引用。因此，Java 对象无法进行拷贝，只能进行修改。另外，Java 对象永远不会消失。clear 函数只从工作区中删除了这些对象的引用，而这些对象本身仍旧存在。Java 的 setVisible 方法仅仅使对象隐藏起来。Java 的 dispose 方法仅仅删除了对象的一些属性，但对象本身依旧存在。最后，如果用户通过退出工作区和使用 clear 命令丢弃了对 Java 对象的引用，那么这些引用也就不复存在了。

在本章最后，有一点提醒大家：Matlab 7 已经大大扩展了 Matlab 对 Java 的支持，相信在以后的版本中，这种扩展还会更加广泛。例如，Matlab 的句柄图形对象和 Java 的图形类和图形对象很可能会进行结合；Matlab 将来很可能会向基于 Java 集成开发环境的方向发展。总之，在将来 Matlab 的开发演进中，Java 将扮演越来越重要的角色。

Chapter 36

Windows 应用程序集成

尽管 C 或 FORTRAN 是目前大部分程序员使用的通用编程语言，但用户也并不是总是需要使用 C 或 FORTRAN 语言来编写程序实现数据之间的通信。UNIX 和 Linux 操作系统使用标准的管道（pipe，即一个程序的标准输出直接进入另一个程序的标准输入）来连接应用程序。而 Microsoft Windows 操作系统则使用动态数据交换（DDE）协议来完成应用程序连接。DDE 允许应用程序可以直接和其他程序进行数据交换或向其他程序发送指令。对象链接和嵌入（OLE）就是基于 DDE 和 Visual Basic 文件扩展名（VBX）建立的一个组件框架。之后，与图形用户接口（GUI）和 Internet 相关的那部分 OLE 又被命名为 ActiveX。随后，Microsoft 又将整个组件框架命名为组件对象模型，即 COM。

一个 COM 对象是一个组件对象类的实例，该实例运行于一个服务器应用程序，却被一个客户端应用程序控制。COM 将所有的数据和方法都封装在一个对象中，并使用接口来访问这些数据和方法。因此，接口就是指向一个对象的方法的指针。COM 对象一般具有多种接口。要想了解某一特定组件的各个接口的详细信息，请读者参考组件开发商提供的帮助文档。

许多商业应用程序都支持 COM 对象、ActiveX 或 DDE，以便于和 Windows 操作系统中的其他应用程序进行交互。Matlab 也提供了对 COM 对象、ActiveX 控件和 DDE 的支持，不过功能有限。Matlab 可以创建 COM 对象来对其他应用程序进行控制，并且也可以作为一个 COM 服务器来响应来自其他应用程序的请求。目前，COM 和 DDE 只能用于基于 Win32 的 Microsoft Windows 操作系统，如 Windows 98、Windows 2000 或 Windows XP。

36.1 COM 对象：客户/服务器通信

COM 协议规定了一个对象模型和一组标准接口与定制接口的集合，用于定义每个对象的方法、属性和事件。方法是可以在对象上执行，或由对象执行的操作（类似于 Matlab 的函数和对象的方法），属性是一系列决定对象状态的变量（类似于句柄图形属性），事件是由于对象的状态被改变而产生的通知指示，通常会触发某些操作（类似于句柄图形或 Java 的回调）。

支持 COM 的应用程序也支持服务器函数（即响应客户端请求的应用程序）、客户端函

数（即向服务器发出请求的应用程序），或同时支持两者。ActiveX 控件是一个可以被 COM 客户端应用程序集成到控制容器（例如 Matlab 图形窗口）中的对象，它通常被一个 COM 服务器应用程序用于初始化一次操作。COM 服务器是一系列能够被一个 COM 客户端或一个 ActiveX 控件控制的应用程序。

Matlab 对 COM 的支持

Matlab 只对有限的 COM 接口提供了支持。通过与其他应用程序交换数据并在 Matlab 工作区中执行命令，这些接口可以作为 Automation 服务器来使用。另外，通过和其他应用程序交换数据，并使用 Matlab 命令行或 M 文件控制 Automation 服务器的应用程序（例如 Microsoft Word、Excel 和 PowerPoint），这些接口还可以作为 Automation 客户端使用。从物理意义上讲，Matlab 是无法嵌入到其他应用程序的，但它可以通过将 Automation 服务器控件嵌入到其图形窗口中（类似于使用 uicontrol），使之可以作为一个控件容器使用。

Matlab 作为客户端

actxserver 函数可以创建一个 COM 对象，并打开一个与 COM Automation 服务器的连接。同样，actxcontrol 函数可以创建一个控件，该控件可以被嵌入到 Matlab 容器（例如一个图形窗口）中，用于控制一个 COM Automation 服务器。下表给出了用于操作 COM 对象的函数（方法）：

函数/方法	功能描述
actxserver	创建一个 COM Automation 服务器
actxcontrol	创建一个 ActiveX 控件对象并将其嵌入到 Matlab 图形窗口中
actxcontrollist	列出当前安装的所有 ActiveX 控件
fieldnames	返回一个对象的属性名
interfaces	列出一个 COM 服务器的定制接口
methods	列出控件或服务器的所有公有方法
events	显示一个控件可以触发的事件列表
eventlisteners	列出附加在受听者（listener）上的事件列表
addproperty	将一个定制属性添加到一个对象
deleteproperty	把一个定制属性从一个对象删除
invoke	调用接口或对象上的一个方法，或显示方法列表
registerevent	将一个事件句柄注册到一个控件上，用于处理某一指定事件
unregisterevent	将一个事件句柄从一个控件上注销
isa	判断是否是给定的 Matlab 或 Java 类
iscom	判断是否是一个 COM 对象
isevent	判断是否是一个事件
isinterface	判断是否是一个 COM 接口
ismethod	判断是否是一个对象的方法
isprop	判断是否是一个对象的属性

（续表）

函数/方法	功能描述
inspect	显示一个图形用户接口（GUI），用于列出和修改属性值
methodsview	显示一个图形用户接口（GUI），用于列出一个对象的方法信息
actxcontrolselect	显示一个图形用户接口（GUI），用于创建一个 ActiveX 控件
class	创建一个对象，或返回一个对象的类
set	设置对象或接口上的一个属性的值
get	从接口获得一个属性值，或显示属性列表
propedit	显示一个控件的内置属性页（如果该属性页存在）
release	释放一个接口
delete	删除一个 COM 控件对象和它所有的接口
load	从一个文件加载控件的属性状态
save	将控件的属性状态存入一个文件
move	移动或缩放控件，并返回新位置的值

actxcontrol、actxserver、get 和 invoke 函数都可以返回 COM 对象（或 COM 对象的新接口——从本质上讲，它用于分隔不同对象间的通信信道），这些 COM 对象可以使用其他 COM 函数或方法来控制或查询。release 函数用于释放 COM 对象或接口不再需要的资源。delete 函数用于关闭所有连接，并释放所有服务器或控制对象不再需要的资源。

使用 help 命令可以获取上述对象方法的帮助文本。例如，下面的代码用于获取 COM 对象的 delete 方法的帮助文本：

```
>> help COM/delete
```

使用 get(axhandle)和 axhandle.get 语法可以获得 COM 对象的属性列表，其中，axhandle 是由 actxserver 或 actxcontrol 方法获得的 COM 对象句柄。例如，要获得 MS PowerPoint COM 服务器的属性列表，可以使用下面的代码：

```
>> ppapp = actxserver('powerpoint.application');

ppapp =
      COM.powerpoint_application

>> get(ppapp)

          Presentations: [1x1 Interface.Microsoft_PowerPoint_9.0_Object_Library.Presentations]
                Windows: [1x1 Interface.Microsoft_PowerPoint_9.0_Object_Library.DocumentWindows]
           ActiveWindow: [1x203 char]
     ActivePresentation: [1x196 char]
       SlideShowWindows: [1x1 Interface.Microsoft_PowerPoint_9.0_Object_Library.SlideShowWindows]
            CommandBars: [1x1 Interface.Microsoft_Office_9.0_Object_Library._CommandBars]
                   Path: 'C:\Program Files\Microsoft Office\Office'
                   Name: 'Microsoft PowerPoint'
                Caption: 'Microsoft PowerPoint'
              Assistant: [1x1 Interface.Microsoft_Office_9.0_Object_Library.Assistant]
             FileSearch: [1x1 Interface.Microsoft_Office_9.0_Object_Library.FileSearch]
               FileFind: [1x51 char]
```

```
                Build: '6620'
              Version: '9.0'
      OperatingSystem: 'Windows (32-bit) 5.00'
        ActivePrinter: 'Xcolor'
              Creator: 1.3479e+009
               AddIns: [1x1 Interface.Microsoft_PowerPoint_9.0_Object_Library.AddIns]
                  VBE: [1x1 Interface.Microsoft_Visual_Basic_for_Applications_Extensibility_5.3.VBE]
                 Left: 99
                  Top: 99
                Width: 720
               Height: 546.7500
          WindowState: 'ppWindowNormal'
              Visible: 'msoFalse'
               Active: 'msoFalse'
         AnswerWizard: [1x1 Interface.Microsoft_Office_9.0_Object_Library.AnswerWizard]
            COMAddIns: [1x1 Interface.Microsoft_Office_9.0_Object_Library.COMAddIns]
          ProductCode: '{00010409-78E1-11D2-B60F-0060774499C8E7}'
    DefaultWebOptions: [1x1 Interface.Microsoft_PowerPoint_9.0_Object_Library.DefaultWebOptions]
     LanguageSettings: [1x1 Interface.Microsoft_Office_9.0_Object_Library.LanguageSettings]
 ShowWindowsInTaskbar: 'msoTrue'
       FeatureInstall: 'msoFeatureInstallNone'
```

使用 invoke 函数可以返回 ppapp 对象的方法列表，以及方法的调用语法，如下所示：

```
>> invoke(ppapp)
   Activate = Void Activate(handle)
   Help = Void Help(handle,Variant(Optional))
   Quit = Void Quit(handle)
   Run = Variant Run(handle,string,SafeArray(Variant))
```

使用 methods 函数，可以获取 COM.powerpoint_application 类对象的所有方法以及继承的方法，如下所示：

```
>> ppapp.methods
Methods for class COM.powerpoint_application:
activate          eq            load          str2double
help              eval          move          str2num
quit              evalc         ne            strcmp
run               events        permute       strcmpi
addproperty       fieldnames    propedit      strmatch
char              findstr       release       strncmp
ctranspose        get           reshape       strncmpi
delete            interfaces    save          transpose
deleteproperty    intersect     send          tril
diag              invoke        set           triu
disp              isletter      setdiff       unique
display           ismember      setxor
double            isspace       sort
```

使用 filenames 函数可以返回包含 ppapp 对象的公有域名称的单元数组，如下所示：

```
>> ppapp.fieldnames
ans =
    'Presentations'
```

```
'Windows'
'ActiveWindow'
'ActivePresentation'
'SlideShowWindows'
'CommandBars'
'Path'
'Name'
'Caption'
'Assistant'
'FileSearch'
'FileFind'
'Build'
'Version'
'OperatingSystem'
'ActivePrinter'
'Creator'
'AddIns'
'VBE'
'Left'
'Top'
'Width'
'Height'
'WindowState'
'Visible'
'Active'
'AnswerWizard'
'COMAddIns'
'ProductCode'
'DefaultWebOptions'
'LanguageSettings'
'ShowWindowsInTaskbar'
'FeatureInstall'
```

最后，使用 events 函数可以列出 ppapp 对象能够触发的所有事件。如下所示（列表中没有区分注册和非注册事件，并且同时给出了在调用事件句柄时需要执行的函数的原型）：

```
>> ppapp.events
    WindowSelectionChange = void WindowSelectionChange(handle Sel)
    W WindowBeforeRightClick = void WindowBeforeRightClick(handle Sel, bool Cancel)
    W WindowBeforeDoubleClick = void WindowBeforeDoubleClick(handle Sel, bool Cancel)
    PresentationClose = void PresentationClose(handle Pres)
    PresentationSave = void PresentationSave(handle Pres)
    PresentationOpen = void PresentationOpen(handle Pres)
    NewPresentation = void NewPresentation(handle Pres)
    PresentationNewSlide = void PresentationNewSlide(handle Sld)
    WindowActivate = void WindowActivate(handle Pres, handle Wn)
    WindowDeactivate = void WindowDeactivate(handle Pres, handle Wn)
    SlideShowBegin = void SlideShowBegin(handle Wn)
    SlideShowNextBuild = void SlideShowNextBuild(handle Wn)
    SlideShowNextSlide = void SlideShowNextSlide(handle Wn)
    SlideShowEnd = void SlideShowEnd(handle Pres)
```

```
PresentationPrint = void PresentationPrint(handle Pres)
AfterNewPresentation = void AfterNewPresentation(handle Pres)
AfterPresentationOpen = void AfterPresentationOpen(handle Pres)
```

根据上面所显示的各个列表，下面我们通过几个具体的例子演示 Matlab 中 COM 和 ActiveX 的特性。

Matlab 作客户端的例子

本例将 Matlab 作为 COM 客户端，将 Microsoft Word 作为 COM Automation 服务器。Matlab 的 M 文件函数 wordfig 首先将当前（或指定）的图形窗口中的内容拷贝到剪贴板上，然后启动一个 Microsoft Word 应用程序服务器实例，之后使用 uiputfile 对话框打开一个指定文件名的文档（或创建一个新的文档），之后在文档的末尾粘贴图像，最后关闭文档，退出 Word 应用程序并删除 COM 服务器对象。下面是 wordfig 函数的具体代码：

```
function wordfig(filespec,popt)
%WORDFIG Open a MSWord document and paste the current figure into it.
% WORDFIG Paste the current Figure window into a word document.
% WORDFIG(FILESPEC) Paste the current Figure window into the document
%    named FILESPEC. Use the complete path to the document if necessary.
% WORDFIG(FILESPEC,POPT) Paste a Figure window into the document
%    FILESPEC using the print options POPT (e.g. -f2 to select Figure
%    window #2).
% If the FILESPEC argument is missing or empty, the uiput file dialog box
%    is used to select a file name.

% Create or verify a valid file name.
if nargin < 1 | isempty(filespec) | ~ischar(filespec)
  [fname,dname]=uiputfile('*.doc','Modify or create the file:');
  if isequal(fname,0), return, end
  filespec=fullfile(dname,fname);
end
[dname,fname,fext]=fileparts(filespec);
if isempty(dname), dname=pwd; end
if isempty(fext), fext='.doc'; end
filespec=fullfile(dname,[fname,fext]);

% Copy the current Figure window onto the clipboard.
if nargin < 2
  print('-dmeta');
else
  print('-dmeta',popt);
end

% Start a session with MSWord.
wrd=actxserver('word.application');
wrd.Visible=1;  % Watch the action...

% Open or create a document.
if ~exist(filespec,'file')
  doc=invoke(wrd.Documents,'Add');
```

```
else
  doc=invoke(wrd.Documents,'Open',filespec);
end

% Insert some text at the end of the document.
myrange=doc.Content;
myrange.InsertParagraphAfter;
invoke(myrange,'InsertAfter','---Figure Top Caption Goes Here---');
myrange.InsertParagraphAfter;
% Paste AFTER the existing text.
invoke(myrange,'Collapse',0);
% Paste with "Picture Format" (1) and "Float Over Text" (3) options.
invoke(myrange,'PasteSpecial',0,0,1,0,3);
myrange.InsertParagraphAfter;
invoke(myrange,'InsertAfter','---Figure Bottom Caption Goes Here---');
myrange.InsertParagraphAfter;

% Save and close the document.
if ~exist(filespec,'file')
  invoke(doc,'SaveAs',filespec,1);
else
  doc.Save;
end
doc.Close;

% Quit Word and close the server connection.
wrd.Quit;
delete(wrd);
return
```

下面的例子将 Matlab 作为 COM 客户端，将 Microsoft PowerPoint 作为 COM Automation 服务器。Matlab 的 M 文件函数 addslide 首先启动一个 Microsoft PowerPoint 应用程序服务器实例，然后使用 uiputfile 对话框打开一个指定文件名的演示文档（或创建一个新的演示文档），之后根据每一个有效的变量参数（可以是字符串、字符串单元数组、二维数值数组、图形句柄等），在文档的末尾插入一张新的幻灯片，之后，将演示文档保存到一个文件并关闭该文档，退出 PowerPoint 应用程序并删除 COM 服务器对象。下面是 addslide 函数的具体代码：

```
function addslide(filespec,varargin)
%ADDSLIDE Insert variables into a PowerPoint presentation.
%
% ADDSLIDE(filename,var1,var2,...) Insert variables into
%   new slides at the end of a PowerPoint presentation.
%
%   An invalid or empty filename or a new filename will
%   bring up a dialog box to enable the user to select an
%   existing file or create a new file.
%
```

```
%   Inputs can be strings, cell arrays of strings, 2D numeric
%   arrays, and figure handles. Variable sizes should be limited
%   to avoid overfilling the slides.

% Check for missing arguments and initialize variables
narg=nargin;
if narg < 2, error('Missing input arguments.'); end
nl=char(13);

% Make sure we have a valid file name
if ischar(filespec) & exist(filespec,'file')  % a real file
  fullname=filespec;
else
  if ischar(filespec) & ~isempty(filespec)    % use as default file
    [dname,fname,fext]=fileparts(filespec);
    if isempty(fext), fext='.ppt'; end
    defname=fullfile(dname,[fname fext]);
  else
    defname='*.ppt';                          % use file filter
  end
  [fname,dname]=uiputfile(defname,'Modify or create the file:');
  if isequal(fname,0), return, end
  fullname=fullfile(dname,fname);
end

% Start a session with PowerPoint.
ppapp=actxserver('powerpoint.application');
ppapp.Visible=1;                          % Watch the action...

% Open or create a presentation
if ~exist(fullname,'file')
  ppres=invoke(ppapp.Presentations,'Add');
else
  ppres=invoke(ppapp.Presentations,'Open',fullname);
end

% Process the rest of the arguments
for idx=2:narg
  arg=varargin{idx-1};

  if ishandle(arg) & (floor(arg)==arg)   % figure handle
    ppslide=ppres.Slides.Add(ppres.Slides.Count+1,'ppLayoutBlank');
    print('-dmeta',sprintf('-f%d',arg)); % copy plot to clipboard
    ppfig=ppslide.Shapes.Paste;          % paste the clip into the slide
    ppfig.Align(1,-1);                   % center the plot
    ppfig.IncrementTop(200);             % lower the plot
    ppfig.ScaleWidth(1.5,0,1);           % scale width from middle

  elseif isstr(arg)                      % text string
    if length(arg) > 275, warning('possible overfull slide'); end
    ppslide=ppres.Slides.Add(ppres.Slides.Count+1,'ppLayoutTitleOnly');
    ppslide.Shapes.Title.TextFrame.AutoSize=1;
```

```
    ppslide.Shapes.Title.TextFrame.VerticalAnchor=1;
    ppslide.Shapes.Title.TextFrame.TextRange.Text=arg;

  elseif iscellstr(arg)                 % cell array of strings
    ppslide=ppres.Slides.Add(ppres.Slides.Count+1,'ppLayoutTitleOnly');
    str='';
    for idx2=1:length(arg)
      str=[str,arg{idx2},nl];
    end
    if (length(arg) > 5) | (length(str) > 275)
      warning('possible overfull slide');
    end
    ppslide.Shapes.Title.TextFrame.AutoSize=1;
    ppslide.Shapes.Title.TextFrame.VerticalAnchor=1;
    ppslide.Shapes.Title.TextFrame.TextRange.Text=str(1:end-1);

  elseif isnumeric(arg) & ~all(ishandle(arg(:))) & (ndims(arg) <= 2)
    ppslide=ppres.Slides.Add(ppres.Slides.Count+1,'ppLayoutTitleOnly');
    [r,c]=size(arg);
    if numel(arg) > 20, warning('possible overfull slide'); end
    rs=int2str(r); cs=int2str(c);
    str=[sprintf('Array %s', inputname(idx)),nl,nl];
    for idx2=1:r
        str=[str, sprintf('    %5.5f',arg(idx2,:)), nl];
    end
    ppslide.Shapes.Title.TextFrame.AutoSize=1;
    ppslide.Shapes.Title.TextFrame.VerticalAnchor=1;
    ppslide.Shapes.Title.TextFrame.TextRange.Text=str;
  else
    warning([inputname(idx),'is not an accepted input and will
           be ignored.'])
  end %if
end %for

% Save the file and exit
ppres.SaveAs(fullname,1,0);          % save as presentation
ppres.Close;                         % close the presentation
ppapp.Quit;                          % quit PPT
delete(ppapp);                       % done with the COM server
return
```

最后一个客户端例子演示了 COM 事件和事件句柄的用法。本例给出的 cdemo 函数将一个 ActiveX 控件嵌入一个图形窗口，并且当该控件的事件被触发时，将执行特定的操作。Matlab 将安装两个非常简单的 ActiveX 控件（mwsamp.ocx 和 mwsamp2.ocx），并将它们的相关类型库文件（mwsamp.tlb 和 mwsamp2.tlb）存储在$Matlab\bin\win32 目录内。Mwsamp 类中包含一个事件（Click 事件）和三个变量（Label、Radius 和 MMPropertyContainer）。其中 MMPropertyContainer 是一个包含各种控件属性的 Java 对象。Mwsamp2 类是 Mwsamp 类的扩展，它在 Mwsamp 的基础上添加了两个事件（DblClick 和 MouseDown 事件）和一个包含接口函数的变量（Ret_IDispatch）。

当一个事件被 ActiveX 控件触发时，Matlab 将向事件句柄传递几个参数，包括对象名、一个数值类型的事件标识符、0 个或多个事件参数、一个包含事件附加信息的事件结构体、以及事件名称（作为最后一个参数）。事件参数的个数和名称会因 ActiveX 控件的不同而不同。

下面给出的 cdemo 函数首先创建一个包含表面（surf）图形的新图形窗口，然后将一个 MWSAMP2 控件嵌入到图形窗口的一角，然后设置控件的属性并退出。当在控件范围内检测到鼠标单击事件时，将再次调用 cdemo 函数以响应回调。当图形窗口关闭后，控件也会随之被删除。下面是 cdemo 函数的具体代码：

```
function cdemo(varargin)
%CDEMO Sample function to manage an ActiveX object.
%  Function to create a sample ActiveX control. The function creates
%  a figure window, adds a nice plot, creates an MWSAMP control,
%  embeds the control in the Figure window, sets the 'Label' and
%  'Radius' properties of the control, and invokes the 'Redraw'
%  method on the control.
%
%  CDEMO is also the event handler for this control. The three events
%  fired by the control are 'Click', 'DblClick', and 'MouseDown'.
%  The event handler changes the text message in the control when
%  a valid event is fired.
%  The control is deleted when the figure window is closed.

% Keep track of a few things between calls.
persistent numclicks h

if nargin == 0      % Initial call -- do the setup.

  % Create a new figure window and draw a nice plot.
  f = figure;
  surf(peaks);
  numclicks=0;

  % Embed an MWSAMP2 ActiveX control in the lower left corner
  % of the Figure window and set the callback to recall this
  % function (cdemo).
  h = actxcontrol('MWSAMP.MwsampCtrl.2',[0 0 90 90],f,'cdemo');

  % Set the initial label and circle radius in the control
  % showing two methods of setting the property values.
  set(h, 'Label', 'Click Here');
  h.Radius=28;

  % Tell the control to redraw itself by invoking the Redraw method.
  invoke(h, 'Redraw');

else    % This part handles the callback. For each valid event
        % detected, the last argument will be a string that
        % resolves to the event name.

  if strcmp(varargin{end},'Click')
```

```
      % Increment the click total.
      numclicks = numclicks + 1;
      h.Label=['Click #',num2str(numclicks)];

    elseif strcmp(varargin{end},'DblClick')
      % Decrement the click total by 2. The first of the pair
      % generated a Click event and incremented by 1. The second
      % click within the time limit generated this DblClick event.
      numclicks = numclicks - 2;
      h.Label=['Click #',num2str(numclicks)];

    elseif strcmp(varargin{end},'MouseDown')
      % Display the x,y coordinates of the mouse pointer.
          h.Label=['(x,y)=(',num2str(varargin{5}),',',
                              num2str(varargin{6}),')'];

    else
      error('Invalid input.');
    end
    % Redraw the control.
    h.Redraw;
  end
```

Matlab 作服务器的例子

当被一个 COM 客户端（例如 Visual Basic、Visual Basic for Application、Visual C++，甚至 Matlab 本身）呼叫时，Matlab 也可以作为 ActiveX Automation 服务器。Microsoft Excel、Microsoft Access 或其他 COM 客户端应用程序也可以作为 Automation 控制器。当 Matlab 作服务器时，客户端应用程序可以启动并停止一个 Matlab 实例，与 Matlab 工作区交换数组，以及在 Matlab 工作区中执行 Matlab 命令。该功能在很大程度上与 Matlab 引擎很相似。实际上，用户使用 C 或 FORTRAN 建立起来的应用都将使用 Matlab 引擎接口，而非 COM 接口。Matlab 的 COM 接口一般是为那些无法将 Matlab 视为服务器进行访问的应用程序保留的。

已注册的 Matlab COM 对象的名称统一为 matlab.application。下表列出了 Matlab 为 COM 客户端应用程序提供的 Automation 方法：

函数/方法	功能描述
Execute	执行包含在的一个字符串参数中的 Matlab 命令
Feval	求一条无法包含在服务器上的单个字符串中的 Matlab 命令的值，例如，那些使用服务器工作区未知的本地变量值的命令
GetCharArray	从服务器获取一个字符串数组
GetFullMatrix	从服务器获取一个矩阵，并返回一个 SAFEARRAY 数据类型的数组
GetWorkspaceData	从服务器工作区获取数据，并返回一个 variant 类型的数组
MaximizeCommandWindow	在 Windows 桌面上显示服务器窗口
MinimizeCommandWindow	使服务器窗口最小化

（续表）

函数/方法	功能描述
PutCharArray	在服务器中保存一个字符串数组
PutFullMatrix	在服务器中保存一个矩阵
PutWorkspaceData	在服务器工作区中保存数据
Quit	退出 Matlab 服务器程序

上表中的方法以及各方法的调用语法也可以通过在 COM.matlab_application 对象上使用 invoke 函数得到。通过将 Matlab 作为一个 COM 客户端使用，可以获取 Matlab COM 服务器的属性列表，下面的代码演示了 Matlab 的这一用法：

```
>> mlapp = actxserver('matlab.application')

mlapp =
    COM.matlab_application

>> get(mlapp)
    visible: 1

>> mlapp.methods
Methods for class COM.matlab_application:

Execute                    evalc          send
GetCharArray               events         set
GetFullMatrix              fieldnames     setdiff
MaximizeCommandWindow      findstr        setxor
MinimizeCommandWindow      get            sort
PutCharArray               interface      str2double
PutFullMatrix              intersect      str2num
Quit                       invoke         strcmp
addproperty                isletter       strcmpi
char                       ismember       strmatch
ctranspose                 isspace        strncmp
delete                     load           strncmpi
deleteproperty             move           transpose
diag                       ne             tril
disp                       permute        triu
display                    propedit       unique
double                     release
eq                         reshape
eval                       save

>> mlapp.filenames
ans =
    'visible'
```

注意：Matlab COM 服务器不支持任何事件，如下所示：

```
>> mlapp.events
```

示例讲解

本例是一个 Excel 宏的例子。该宏首先启动 Matlab 并将其作为 Automation 服务器，然后将 B3:E8 单元格中的内容传递给一个 Matlab 数组，之后对数组的内容进行乘方运算，并将结果插入到 Excel 表格中的 B12:E17 单元格中。同时，该宏还将从 B1 单元格中获得的字符串值传递给 Matlab，然后在字符串后面追加“squared(in Matlab)”字符串，之后将结果插入到 B10 单元格中。最后，关闭 Matlab 服务器。下面是本例函数的具体代码：

```
Sub Square()
'
' Square Macro
' Square the contents of cells B3:E8 and place the result in B12:E17.
' Also append the phrase " squared (in MATLAB)" to a string from B1
' and place the result in B10.

' First define the variables.
Dim MatLab As Object
Dim Result, NewString As String
Dim MReal(6, 4) As Double
Dim MImag() As Double
Dim RealValue As Double
Dim i, j As Integer

' Invoke MATLAB.
Set MatLab = CreateObject("Matlab.Application")

' Fill the Mreal array with data from the sheet.
For i = 0 To 5
    For j = 0 To 3
      Real(i,j)=ActiveSheet.Range(Cells(i+3,j+2),Cells(i+3,j+2)).Value
    Next j
Next i

' Send the string and data from the spreadsheet to MATLAB.
Call MatLab.PutCharArray("instr", "base",
                              ActiveSheet.Range("B1:B1").Value)
Call MatLab.PutFullMatrix("a", "base", MReal, MImag)

' Send MATLAB some commands to execute.
Result = MatLab.Execute("b=a.^2;")
Result = MatLab.Execute("outstr=[instr, ' squared (in MATLAB)']")

' Retrieve the results and stuff them into spreadsheet cells.
Call MatLab.GetFullMatrix("b","base",MReal,MImag)
ActiveSheet.Range("B12:E17").Value = MReal
ActiveSheet.Range("B10:B10").Value=MatLab.GetCharArray("outstr","base")
'
End Sub
```

假如我们给定下面的 Excel 电子表格：

	输入数组的值			
	1	2	3	4
	2	3	4	5
	3	4	5	6
	4	5	6	7
	5	6	7	8
	6	7	8	9

运行上述 sub square()宏后，电子表格变为：

	输入数组的值			
	1	2	3	4
	2	3	4	5
	3	4	5	6
	4	5	6	7
	5	6	7	8
	6	7	8	9
	输入数组的平方值（在 Matlab 中显示）			
	1	4	9	16
	4	9	16	25
	9	16	25	36
	16	25	36	49
	25	36	49	64
	36	49	64	81

当然，上述只是一个实用价值有限的简单示例，但它是我们进一步开发更复杂的实例的基础。

> 创建 VBA 宏的一个简单方法是记录一个新的宏，并在此基础上编辑 VBA 代码。例如，打开 Excel，并选择 Tools/Macro/Record New Macro...菜单项，然后在打开的对话框中为宏取一个名字（比如上例中宏的名字为 square），完成后单击 OK 按钮。然后，单击下一个对话框中的 Stop Recording 按钮。这时，一个新的宏就产生了。之后，选择 Tools/Macro/Macros...菜单项，并在出现的对话框中单击 Edit 按钮。此时，用户就可以编辑宏，保存编辑结果，以及运行该宏了。

除了前面所演示的 Matlab 的 get 和 invoke 方法外，Visual Basic 环境也提供了一些工具帮助用户选择适当的对象和方法与其他 COM 应用程序进行通信。利用对象浏览器（Object Browser）可以判断一个对象库（如 Excel 库或 MSForms 库）中都有哪些对象，可以对对象及其成员（包括属性、事件和函数）进行搜索，另外还可以获取上下文帮助。

36.2 动态数据交换

在 COM 的客户端/服务器功能出现之前，Microsoft 已经建立了一种主要用于点对点通信的内部应用程序协议，称为动态数据交换（Dynamic Data Exchange，DDE）。DDE 通过使用 Windows 的剪贴板，使两个协同工作的应用程序互相交换数据和命令。一个应用程序可以在另一个应用程序的某些数据发生改变时，注册一个更新通告请求。现在，DDE 的功能已经合并到了更加灵活的 COM 协议中。虽然一些已有的代码仍旧支持 DDE，但是新代码一般都使用 COM。

应用程序之间的 DDE 连接通常被称为会话（conversation）。每个应用程序都有一个惟一的服务名（Service Name）作为识别号。每个会话都通过应用程序双方都知道的一个服务名和一个主题（Topic）来标识。每个应用程序都至少会支持一个系统主题（System Topic），其中大多数应用程序都支持一个或多个其他主题。在会话期间，发出 DDE 请求的应用程序被指定为客户端，对请求做出应答的应用程序被指定为服务器。

一个典型的 DDE 会话首先从应用程序 A 向应用程序 B 的实例发出一个会话请求开始。一个会话通常由一个“服务名/主题”对组成。如果应用程序 B（由服务名指定）能够识别会话中的主题，就会与应用程序 A 建立一个会话。会话元素就是一些使用 Windows 剪贴板在应用程序间传递的条目（Item）。所有应用程序都支持文本格式的数据交换，有些应用程序还支持其他数据格式，如图形的 Bitmap 和 MetaFilePict 格式，Excel 电子表格数据的 XLT 表格格式等。Matlab 作为 DDE 客户端只支持文本格式，但作为 DDE 服务器可以支持文本、XLT 表格和 MetaFilePict 格式。

一个 COM 连接请求可以激活一个应用程序实例（即运行该应用程序）来响应客户端的请求，但是 DDE 会话只能在已经运行的应用程序之间建立。

Matlab 用作 DDE 客户端

下表给出了 Matlab 支持的 DDE 函数：

函数	说明
ddeadv	在 Matlab 和 DDE 服务器应用程序之间建立一个咨询连接
ddeexec	向 DDE 服务器应用程序发送一个执行字符串（即命令）
ddeinit	在 Matlab 和另一个应用程序之间建立一个 DDE 会话
ddepoke	从 Matlab 向一个 DDE 服务器应用程序发送数据
ddereq	请求 DDE 服务器应用程序发送数据
ddeterm	结束 Matlab 和服务器应用程序之间的 DDE 会话
ddeunadv	释放 Matlab 和 DDE 服务器应用程序之间的咨询连接

以上函数通常用来管理 Matlab 与其他作为 DDE 服务器的应用程序之间的 DDE 连接。这些 DDE 服务器应用程序包括 Microsoft Excel、Word、Access、PowerPoint，甚至是 Matlab 本身。

作为服务器，Microsoft Excel 支持两种类型的主题：系统主题和一个名称主题，其中

名称主题由在 Excel 里打开的工作簿的名称或打开的工作簿中的电子数据表的名称组成。Microsoft Excel 条目可以是对任何单元格的引用：可以是单独的一个单元格，也可以是一系列单元格。Microsoft Word 也支持两种类型的主题：系统主题和一个由 Word 中打开的文档名组成的主题。Microsoft Word 条目可以是已打开文档的任何书签。关于主题和它所支持的条目的详细内容，请参考各 DDE 应用程序的在线帮助或打印文档。

下例演示了 Matlab（客户端）和一个打开的 Excel 副本（服务器）之间的 DDE 对话。首先，我们请求一个到系统主题的连接（每个 DDE 应用程序都支持系统主题），代码如下：

```
>> xls = ddeinit('excel','system')
>> xls =
    4.7836e-299
```

上例返回的值是所建立的会话的句柄。如果连接请求失败，那么返回的句柄值为 0。随后，我们需要使用标准的 SysItems 条目来获得系统主题下可以使用的所有条目的列表。其中第三个参数是一个可选参数，它是用于定义格式的向量。该向量的第一个元素用于指定剪贴板的格式：1 为文本格式——它是 Matlab 作客户端惟一支持的格式；第二个元素用于指定结果的存储方式：0 表示将数据作为数值来存储（默认），1 表示将数据作为字符串来存储。实现上述功能的 Matlab 代码为：

```
>> s = ddereq(xls,'SysItems',[1 1])
s =
SysItems Topics Status Formats Selection Protocols EditEnvItems
```

我们可以使用相同的 ddereq 函数，查询每一个条目的详细内容：

```
>> t = ddereq(xls,'Topics',[1 1])
t =
[Book1]Sheet1  [Book1]Sheet2  [Book1]Sheet3  System

>> t = ddereq(xls,'Status',[1 1])
t =
Ready

>> t = ddereq(xls,'Formats',[1 1])
t =
XlTable Microsoft Excel 8.0 Format Biff4 Biff3 SYLK Wk1 Csv Unicode Text
Text Rich Text Format DIF Bitmap Picture (Enhanced Metafile)
Printer_Picture Screen Picture EMF

>> t = ddereq(xls,'Selection',[1 1])
t =
[Book1]Sheet1!R1C1

>> t = ddereq(xls,'Protocols',[1 1])
t =
StdFileEditing  Embedding

>> t = ddereq(xls, EditenvItems',[1 1])
t =
StdHostNames    StdTargetDevice StdDocDimensions
```

上面的结果显示：打开的Book1 工作簿包含 3 个工作表，分别为 Sheet1、Sheet2、Sheet3（第一条语句）；Excel 已经准备好进行会话（第二条语句）；Excel 支持多种剪贴板格式（第三条语句）；当前选取的单元格位于第一行第一列（第四条语句）；此外还有其他一些信息。Excel 支持系统主题和一个工作表可用的主题，而工作表主题只支持一种条目类型——单元格条目。

要关闭会话连接，可以使用下面的代码：

```
>> status = ddeterm(xls)
status =
    1
```

下面的例子将打开一个与单个工作表的会话。首先，创建一个纵横图，并将其绘制出来：

```
>> m = magic(10);
>> h = surf(m);
```

打开一个与工作表的连接，然后将数据插入单元格中：

```
>> xl = ddeinit('excel','Sheet1');
>> range = 'r1c1:r10c10';
>> status = ddepoke(xl,range,m);
```

现在，在电子表格数据和 Matlab 之间建立一个咨询链接（Advisory Link），以便在单元格内的数据发生改变时触发一个回调：

```
>> status = ddeadv(xl,range,'disp(''Spreadsheet data change alert!'')');
```

其中第三个参数就是传递给 eval 函数的回调。此时，一旦电子表格中的一个或多个单元格内容发生了改变，就会在命令窗口显示一条消息：Spreadsheet data change alert。

现在，关闭链接，然后打开一个新的链接。此时，如果电子表格发生改变，那么链接将把表格中的数据传递给 Matlab 变量 z，然后，回调使用新的数据更新绘制的图形。其中第四个参数将通知服务器在向 Matlab 发送信号执行回调的同时，还需要将数据发回。实现上述功能的代码如下：

```
>> status = ddeunadv(xl,range1);
>> status = ddeadv(xl,range,'set(h,''ZData'',z);
            set(h,''CData'',z);','z');
```

用户可以自己修改一些数据，然后观察对绘图有何影响。

用户可以使用 ddeexec 函数向 DDE 服务器传送一条可以执行的命令。例如，下面的代码将第三行第五列的单元格设置为当前的单元格：

```
>> status = ddeexec(xl,'[formula.goto(''r3c5'')]');
```

执行完任务后，用户可以使用下面的代码关闭链接，结束会话：

```
>> status = ddeunadv(xl,range1);
>> status = ddeterm(xl);
```

Matlab 用作 DDE 服务器

Matlab 作为 DDE 服务器时，DDE 的服务名就叫做 Matlab。Matlab DDE 服务器支持两类主题：系统主题和引擎主题。下表给出了这两类主题中的有效条目：

主题	条目	描述
System	SysItems	系统主题下支持的条目列表，以制表符分隔（包括 SysItems，Format 和 Topics）
	Format	所有支持格式的列表，以制表符分隔（包括 Text，MetaFilePict 和 XLTable）
	Topics	所有支持主题的列表，以制表符分隔（包括 System 和 Engine）
Engine	EngEvalString	发送要执行的命令时的条目名称（当客户端调用语法需要时提供）
	EngStringResult	DDE 的 execute 命令返回的字符串结果
	EngFigureResult	DDE 的 execute 命令返回的图形结果
	<matrixname>	将被创建、更新或返回的矩阵的名称

DDE 客户端应用程序通常要使用服务名 Matlab 和系统主题（或引擎主题）来打开一个与 Matlab 的会话。Matlab 引擎主题支持以下 3 种客户端操作：向 Matlab 发送数据，从 Matlab 接收数据，向 Matlab 发送一个要执行的命令。

客户端可以使用 DDE 的 poke 操作来发送数据，发送的条目是要创建或更新的矩阵名称，发送数据的格式是文本格式或 XLT 表格格式。其中，文本格式必须是以制表符分隔的 ASCII 格式，换行符是 CR/LF 格式；而 XLT 表格格式可以是任何的 Excel 二进制数据格式。客户端可以用 3 种方式使用 DDE 的 request 操作从 Matlab 中获取数据：①如果要获取数组中的数据内容，则条目应是发送矩阵的名称，数据格式应为文本格式或 XLT 表格格式。②如果要获取前一条 Matlab 命令的文本输出，则条目应是 EngStringResult，数据格式应为文本格式。③如果要获取前一条 Matlab 命令的图形输出，则条目应为 EngFigureResult，数据格式应为 MetaFilePict。对于那些只能使用 request 操作接收文本的客户端，Matlab 将以文本格式指定 EngFigureResult 条目。此时，所得到的结果将是一个状态字符串。如果该字符串为“Yes”，则表明客户端可以在剪贴板上获得所需要的图形，如果为“No”，则表明操作失败。

客户端可以使用 DDE 的 excute 操作向 Matlab 发送命令，发送的条目为 EngEvalString，命令格式为文本格式。有些客户端可能会跳过条目，直接发送命令。Matlab 同时支持上述两种命令发送方式。

36.3　Matlab 记事本

Matlab 记事本是创建报表和其他文档的简易工具，它可以将 Matlab 命令、命令的输出以及图形嵌入到报表中。Matlab 记事本使得用户能在 Microsoft Word 文档中直接访问 Matlab（此时将 Matlab 视为一个 COM Automation 服务器），而无需使用 C 或 FORTRAN 编写任何附加的程序。Microsoft Word 是 Matlab 记事本惟一支持的字处理软件。用户需要单独购买并将其安装到带有 Windows 或 Macintosh 操作系统的计算机上。通过记事本，用户可以

创建带有 Matlab 命令行的报表，并且命令行执行的结果会自动嵌入到文档中。记事本通常由一系列预定义的 Microsoft Word 宏和 COM 控件来执行，这些宏和控件使用户可以在一个称为 M-book 的 Word 文档中建立一个与 Matlab 执行周期的会话连接。当用户在一个安装有 Microsoft Word 或 Office 的 Win32 PC 平台上安装 Matlab 时，上述功能模块都会自动安装进来。

记事本将使用一个 Microsoft Word 模板将控件嵌入到用户的文档中。命令 notebook –setup 用于对记事本进行初始化。执行该命令后，系统将会提示用户选择需要使用的 Word 版本（包括 Word97、Word2000 或 Word2002）。然后，会询问 Microsoft Word 程序的安装位置，并安装模板文件的正确版本。如果找不到可执行文件，则安装脚本会使用一个文件请求程序询问 Microsoft Word 程序的位置，并使用另一个文件请求程序询问模板文件（例如 normal.dot）的位置。确认无误后，正确的模板文件（此处为 m-book.dot）将被拷贝到相应的目录中。当用户下次使用 notebook 命令时，Microsoft Word 将使用 m-book.dot 模板打开。如果在打开 Microsoft Word 时，用户提供了文件名参数，那么 Word 会打开相应的文件；如果没有提供参数，那么 Word 会打开一个新建的 M-book 文档。

Matlab 7 已不支持基于 Macintosh 操作系统的记事本，也就是说，notebook 命令只能用于 Windows 操作系统。相信在以后的版本中，这一功能还会被添加上。

当用户在 Matlab 中使用 notebook 命令打开一个 M-book，或在 Microsoft Word 中打开一个已有的 M-book 时，就会启动一个新的 Matlab 服务器执行周期，并且会在 Word 应用程序的 Table 菜单旁新增一个 Notebook 菜单。Notebook 菜单包含可以进行如下操作的菜单项：定义和取消定义 Matlab 输入语句（或单元）；对这些单元进行分组和拆分；定义文档中的计算区域；执行单元；在计算区域或整个 M-book 中进行运算等。当一个输入单元被执行后，该语句将被发送到 Matlab 服务器进行求解，并且将求解的结果（既可以是命令行结果，也可以是一幅图形）插入到 M-book 文档的输入声明之后。另外，用户也可以在 Notebook 菜单中对记事本属性进行设置。

> 如果用户不想在 Matlab 中使用 notebook 命令创建新的 M-book 文档，则可以在 Word 中创建。首先，启动 Word，选择 Tools 菜单中的 Templates and Add-Ins… 菜单项。从弹出的对话框中选择 mbook.dot 模板（选中后会出现一个复选标志），然后关闭对话框。如果 Global template 区域没有 mbook.dot 模板，则单击 Add… 按钮，并选择 mbook.dot 文件将其添加进来。选中 mbook.dot 模板后，就会出现 Notebook 菜单，文档也会变成相应的 M-book 文档。最后保存该文档并关闭 Word。

下面我们举一个简单的例子。首先看下面的 M-book 文档：

```
                    Matlab Notebook Example 1

                         Assignment #2

Create a 3-by-3 magic square

     m=magic(3)

square the elements of the magic square

     m2=m.^2

and plot the result

     plot(1:9,m2(:))
```

选中其中一条 Matlab 语句，然后选择 Notebook 菜单中的 Define Input Cell 菜单项。这样就将选中的文本转化为了一个输入单元，即一条可执行的 Matlab 语句。按照上述方式，将其余两条语句也转化为可执行语句，则上面的 M-book 文档将变成如下形式：

```
                    Matlab Notebook Example 1

                         Assignment #2

Create a 3-by-3 magic square

    [m=magic(3) ]

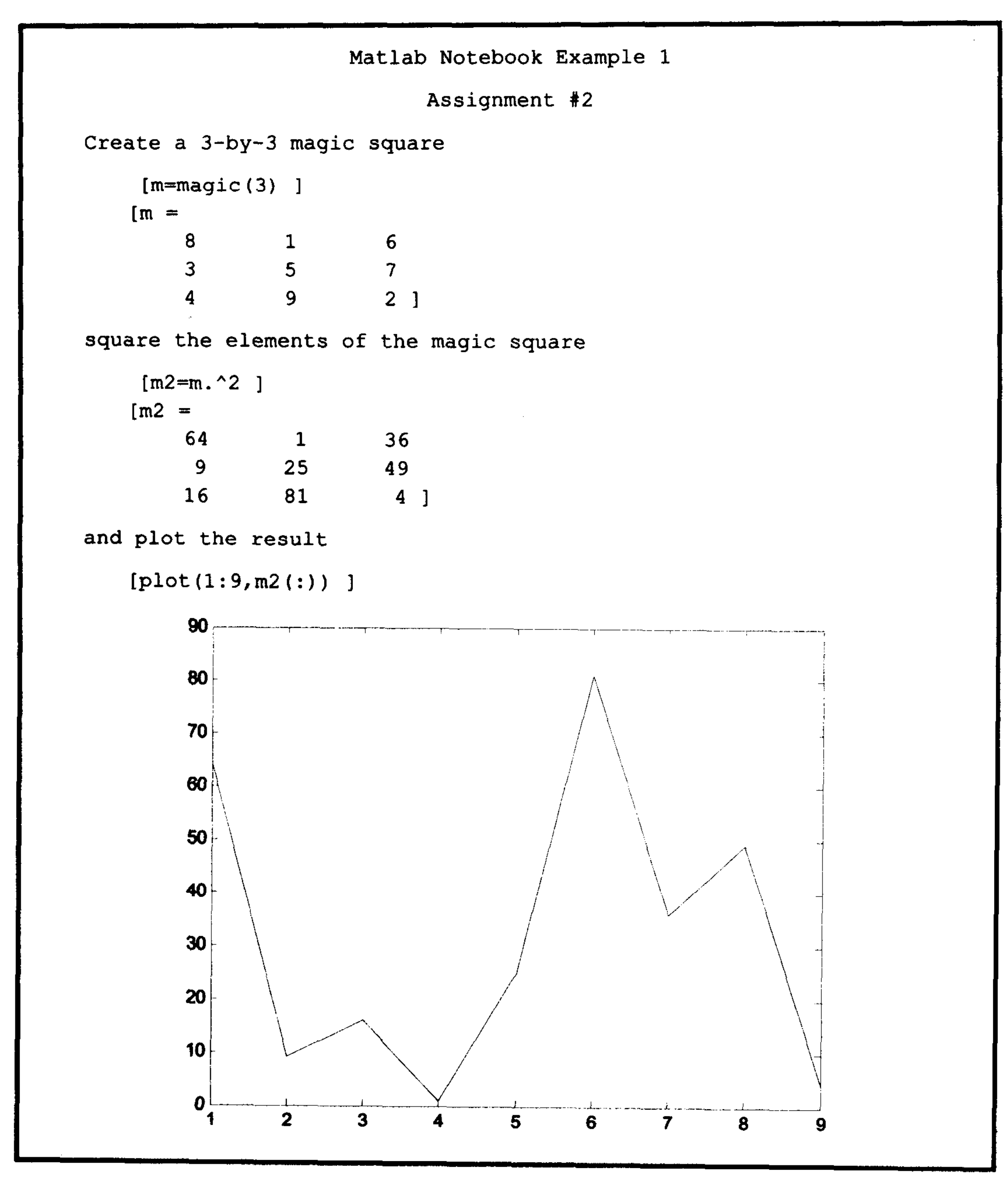
square the elements of the magic square

    [m2=m.^2 ]

and plot the result

    [plot(1:9,m2(:)) ]
```

由上面的 M-book 文档可知，输入单元中的命令文本变为了定点格式的字体，并放入了一对方括号之中（当文档打印时方括号是不显示的）。此时，如果用户选择了 Notebook 菜单下的 Evaluate M-book 菜单项，则应用程序将会连接到一个已经打开的 Matlab 会话执行周期，或启动一个新的 Matlab 执行周期；然后执行 Matlab 语句；最后将结果插入到 M-book 文档中。执行以后的 M-book 文档如下所示：



Matlab Notebook Example 1

Assignment #2

Create a 3-by-3 magic square

```
 [m=magic(3) ]
[m =
     8     1     6
     3     5     7
     4     9     2 ]
```

square the elements of the magic square

```
 [m2=m.^2 ]
[m2 =
    64     1    36
     9    25    49
    16    81     4 ]
```

and plot the result

```
[plot(1:9,m2(:)) ]
```


记事本可以用来创建一个 Matlab 执行周期的注释记录（就像使用 diary 命令为该执行周期创建一个日记一样），也可以用来在报表中插入 Matlab 实例。对 M-book 文档中的输入单元的任何改变都会影响该单元的输出（即执行结果）。因此，从这个角度上讲，M-book 文档属于动态文档。

Word 中打开的所有 M-book 文档都共享同一个单独的 Matlab 实例，并且共享同一个 Matlab 工作区。因此，如果用户打开了多个文档，那么同名变量将在所有 M-book 文档之间共享，也就是说，如果一个 M-book 文档中的变量被修改了，其他 M-book 文档中的同名

变量也将跟着改变。由于用户可以随时定义和取消输入单元和计算区域，因此，上述的共享特性很容易被这些操作所打破。当 Matlab 执行完 M-book 文档中的输入单元或整个 M-book 后，其中的文本输出可以被刷新，但图形输出只能被复制。例如，对于前面的 M-book 文档，如果未清除前一次的单元输出就再次执行 M-book，则该文档会在末尾包含两幅图像。因此，用户应该随时选择 Purge Output Cells 和 Evaluate M-book 菜单项来确保数据的一致性。

36.4 Matlab 中与 COM 有关的工具箱

目前，MathWorks Inc.正在发行一种称为 Excel Link 的附加工具箱产品（需要购买），该产品与 Matlab Notebook 功能类似，只不过它是用于 Microsoft Excel 电子表格的。利用该产品，用户可以在 Excel 中轻松地控制数据交换和 Matlab 命令。如果用户经常需要在 Excel 和 Matlab 之间进行大规模的数据传递，或经常使用 Matlab 对 Excel 数据进行复杂的计算来分析，建议您购买一套 Excel Link 工具箱，这样可以节省不少计算处理时间。

除此以外，用户还可以得到其他一些与 COM 有关的 Matlab 附加工具箱。例如，Matlab COM Builder 工具箱可以用于将 Matlab 算法转化为 COM 对象，以供所有基于 COM 的应用程序调用；Matlab Excel Builder 工具箱可以将复杂的 Matlab 算法转化为独立的 Excel 内插附件；Matlab Report Generator 使用户可以从 Matlab 模型创建标准和定制的报表，并可以使用多种输出格式组织数据，包括 HTML、RTF、XML 和 SGML。以上这些工具箱其实都是根据我们前面讲到的技术和函数创建的函数集。

36.5 小结

由于使用管道连接 UNIX 和 Linux 操作系统上的应用程序的标准输入输出简单易行，一直以来，该技术一直被作为应用程序之间的通信方式。目前，Windows 操作系统上应用程序间的标准化通信协议也开始逐渐被用户接受，并呈迅速发展的趋势。Matlab 支持 Microsoft 的 DDE 和 COM 协议，使得 Matlab 可以作为客户端或服务器与其他支持 DDE 或 COM 的应用程序（如 Microsoft Word、Excel、PowerPoint）进行通信。另外，用户也可以通过一些附加工具箱使 Matlab 和其他 COM 应用程序之间的集成变得轻松自如。

Chapter 37

Matlab 帮助

随着 Matlab 版本的不断演进，Matlab 帮助文档也在逐步改进。最早的时候，用户只能在命令窗口使用 help 和 lookfor 命令查看帮助。随着 Matlab 版本的改进，有些旧的帮助函数已经过时不再使用，而有些则被进行了修改用作它用。本章主要介绍 Matlab 向用户提供帮助的能力，包括如何通过 Internet 查找所需的资源。

37.1 命令窗口帮助

在 Matlab 的图形用户接口（GUI）出现之前，用户只能使用 help 和 lookfor 函数在命令窗口中查看帮助。这两个函数至今仍在使用。例如，下面的代码用于查看 sqrt 函数的帮助文本：

```
>> help sqrt
SQRT Square root.
    SQRT(X) is the square root of the elements of X. Complex
    results are produced if X is not positive.

    See also SQRTM.
```

通常情况下，如果用户知道函数名，但想查看它的输入或输出参数，那么就可以使用 help 函数。在上面所显示的帮助文本中，函数名被大写仅仅是为了突出显示。在函数调用时，Matlab 是区分函数名的大小写的。例如：

```
>> SQRT(2)
??? Undefined command/function 'SQRT'
```

出现了一个错误提示，这是因为用户将 sqrt 函数误写成了大写字母。

如果用户不知道具体的函数名，但知道与该函数相关的某个关键字，则可以使用 lookfor 函数进行查找。例如，如果用户想使用某个与关键字'inverse'有关的函数，可以使用下面的代码进行查找：

```
>> lookfor inverse
INVHILB Inverse Hilbert matrix.
ACOS    Inverse cosine.
```

```
ACOSH   Inverse hyperbolic cosine.
ACOT    Inverse cotangent.
ACOTH   Inverse hyperbolic cotangent.
ACSC    Inverse cosecant.
ACSCH   Inverse hyperbolic cosecant.
ASEC    Inverse secant.
ASECH   Inverse hyperbolic secant.
ASIN    Inverse sine.
ASINH   Inverse hyperbolic sine.
ATAN    Inverse tangent.
ATAN2   Four quadrant inverse tangent.
ATANH   Inverse hyperbolic tangent.
ERFCINV Inverse complementary error function.
ERFINV  Inverse error function.
INV     Matrix inverse.
PINV    Pseudoinverse.
IFFT Inverse discrete Fourier transform.
IFFT2 Two-dimensional inverse discrete Fourier transform.
IFFTN N-dimensional inverse discrete Fourier transform.
IFFTSHIFT Inverse FFT shift.
IPERMUTE Inverse permute array dimensions.
UPDHESS Performs the Inverse Hessian Update.
INVHESS Inverse of an upper Hessenberg matrix.
```

lookfor 函数在执行时将打开 Matlab 搜索路径中的所有 M 函数文件，然后在文件中的第一行注释（即 H1 帮助行）中寻找给定的关键字，最后返回所有匹配的 H1 帮助行。

37.2 帮助浏览器

在较新的 Matlab 版本中，除了 help 和 lookfor 函数外，还提供了相对分离的帮助浏览器或帮助窗口。要打开 Matlab 帮助窗口，用户可以选择 Matlab 桌面上 Window 菜单中的 Help 菜单项，或在 Matlab 命令窗口中直接输入 helpwin、helpdesk 或 doc。帮助窗口不仅用于显示帮助文本，还提供了帮助导航功能。帮助导航提供了 4 个选项卡：Contents、Index、Search 和 Demo。其中，Contents 选项卡中提供了 Matlab 和所有工具箱的在线文档的内容列表；Index 选项卡提供了所有在线帮助条目的索引；Search 选项卡允许用户在在线文档中进行搜索；Demo 选项卡则提供了 Matlab 演示函数的接口。与命令窗口中的纯文本帮助相比，帮助浏览器使用户更容易更方便获取需要的帮助信息，所以用户应该熟练掌握帮助浏览器的使用。

help 函数和 helpwin 函数在显示帮助内容上是等效的，只不过 helpwin 函数将帮助内容显示在一个帮助窗口中，而不是在命令窗口中直接显示。例如，下面的代码将打开一个帮助窗口用于显示 sqrt 函数的帮助文本：

```
>> helpwin mmpad
```

实际上，Matlab 在执行上述代码时，首先打开 mmpad.m 文件，读取帮助文本，然后将文件转换成 HTML 格式，并在帮助窗口中显示该 HTML 文本。在该过程中，大写的函数

都将被转换成小写格式，列在“See also”后面的参考函数都被转换成能够链接到相应函数的 HTML 链接。另外，如果函数还有扩展的联机帮助文档，那么会在帮助文本顶端出现一个到该文档的链接。如果是用户自己创建的工具箱或 M 文件函数集，helpwin 也可以正常显示这些函数的帮助文本。如果 helpwin 函数后面的参数是一个工具箱目录名，那么会在帮助窗口中打开并显示该目录中的 Contents.m 文件。

doc 函数会绕过 M 文件的帮助文本，直接连接到在线帮助文档。例如，下面的代码将显示 print 函数的在线文档：

```
>> doc print
```

在线帮助文档包含了比帮助文本更多更详细的信息。

whatsnew 函数和 whatsnew toolbox 语句用于在帮助窗口中显示 Matlab 或某个选定工具箱的发布信息和最后修改时间。实际上，whatsnew toolbox 语句在后台打开了工具箱的 Readme.m 文件，并在帮助窗口中显示出来。

37.3 Internet 资源

Mathworks Inc.（Matlab 的制造商）的网站是互联网上排名在前 100 名的商业网站，其网址是 http://www.mathworks.com。该网站提供了涵盖 Matlab 的各个方面的信息。由于该网站内容繁多，并且会经常更新，不停地添加新内容、删除旧链接，因此，本书无法也没有必要对网站上的具体内容进行讲解，有兴趣的读者可以到该网站上一饱眼福。不过，Mathworks Inc.的网站上有两个最有用的工具需要提示一下：一个是解决方案搜索引擎（Solution Search Engine），另一个是 Matlab 中心（Matlab Central）。解决方案搜索引擎主要为通用的技术问题提供答案；Matlab 中心则存储了用户和 Matlab 开发商发布的函数 M 文件。用户还可以在该网站上找到对本书的介绍。

如果您正在使用 Matlab，那么无需退出 Matlab 来访问它的网站。Matlab 帮助浏览器本身就可以作为一个简单的网页浏览器使用。当然，用户也可以使用其他的网页浏览器浏览。

除了 Mathworks Inc.网站以外，互联网上的新闻组 comp.soft-sys.matlab 也是一个很热门的 Matlab 论坛。访问该新闻组通常需要软件支持和许可证号，另外，用户也可以通过 Mathworks Inc.网站中的 Matlab 中心登录该新闻组。该新闻组由许多经验丰富的 Matlab 高手定期维护，并对客户提出的问题进行解答。如果您有在在线帮助中无法明确解答的问题，可以将该问题提交到 Matlab 新闻组，在那里或许能得到圆满答复。另外，新闻组还提供了一个 FAQ（常见问题解答）列表，用于回答用户经常遇到的通用问题，用户可以通过下面的网址访问该列表：www.mit.edu/~pwb/cssm/matlab-faq.html。

37.4 本书的帮助

读者在学习本书时，除了可以获取 Matlab 中提供的帮助外，还可以获取本书作者们提供的两个帮助服务。

首先，用户可以访问专为本书建立的一个 Mastering Matlab 网站，地址是：

http://www.eece.maine.edu/mm。在该网站上，用户可以下载用于生成本书中图形的 M 脚本文件。利用这些代码，用户就不用在命令窗口中重复输入了。另外，该网站还有本书中出现的 Java、MEX 和函数 M 文件的代码以供下载。最后，本书的勘误表和互联网上其他与 Matlab 相关的链接也可以在该网站上找到。

其次，本书作者们提供了一个电子邮件地址：mm@eece.maine.edu。读者如果有什么问题和建议，可以向此地址发送邮件。我们热切盼望读者能够向我们反馈本书中出现的错误，帮助我们使下一个版本更加完善。我们也欢迎读者对本书的内容和例子提出问题。当然，您也可以问一些普通的 Matlab 问题，不过，如果您问的问题过于简单，而且可以在 Matlab 的帮助文档、Matlab FAQ 或 Matlab 新闻组中轻易找到答案，我们有可能不会回答您的问题，敬请见谅。

37.5 小结

下表总结了本章所讲的主要内容和一些重要的网址和邮件地址：

条目	描述
help functionname	在命令窗口中显示 functionname 函数的帮助文本
lookfor keyword	在命令窗口中显示所有含有 keyword 关键字的函数及其 H1 帮助行
helpwin functionname	在帮助窗口中显示 functionname 函数的帮助文本
doc functionname	在帮助窗口中显示 functionname 函数的在线帮助
helpbrowser, helpdesk	打开帮助窗口，并显示帮助主页面
whatsnew	显示版本发布信息或工具箱的 Readme.m 文件
http://www.mathworks.com	Mathworks Inc.的网址
comp.soft-sys.matlab	Matlab 新闻组
www.mit.edu/~pwb/cssm/matlab-faq.html	Matlab 新闻组 FAQ
http://www.eece.maine.edu/mm	本书的网址
mm@eece.maine.edu	本书作者的电子邮件地址

Chapter 38

综 合 实 例

俗话说，一图值千字。同样，对于应用软件而言，一例值千言。本书将通过介绍一些扩展实例，作为本书的综合总结。其中的一些例子都涉及到向量化（Vectorization），利用它可以最大化使用数组操作编写代码。另一些例子则涉及到了如何最大化地利用 Matlab 的 JIT 加速器特性提供的优势创建代码。其他一些例子则向读者阐述了一些典型问题的解决办法。在讲解综合实例之前，我们首先向大家介绍一下向量化和 JIT 加速的概念。

38.1 向量化

向量化指编写或修改代码时，用数组操作代替基于数组元素的标量操作。例如，下面的代码：

```
>> for i=1:n
        y(i) = sin(2*pi*i/m);
   end
```

可以被下面的向量化代码取代：

```
>> i = 1:n;
>> y = sin(2*pi*i/m);
```

在上面的两段代码中，前一段 For 循环代码的编程效率远低于后面的向量化代码，这是因为：①从该例的实现角度讲，向量化方法更加简单，For 循环则稍显复杂。②For 循环的执行速度低于向量化方法，这是因为在循环中每次都要为变量 y 重新分配内存。

不过，有向量化方法并不意味着用户就不再需要使用 For 循环。For 循环也有 For 循环的优势和用武之地，要不然，Matlab 也不会将其作为一种基本的控制流程结构提供给用户。通常，如果能充分利用 Matlab 的 JIT 加速特性，For 循环是最佳选择。另外，在以下情况中，也建议用户首选 For 循环：①循环体内的代码总数比较少（尤其是这些代码需要进行大量的实数浮点运算）。②能保证循环体中调用的 M 函数文件最少。③在循环体内，经过第一次循环之后，就不再进行内存的分配和再分配。④使用 For 循环可以避免创建过大的数组，从而使计算机可以以最小延迟时间访问。很明显，最后一种情况不仅与操作系统有关，而且与操作系统内的硬件属性有关。

虽然向量化可以提高 Matlab 的编程效率，但也会带来一些负面影响。主要是向量化的代码比较难于阅读和理解。不过这一点我们从上面的例子中还看不出来，反而觉得向量化的代码更容易阅读一些。但这个例子毕竟只是一个最简单的例子，在大多数情况下，向量化的代码对用户来说都是很不明朗的，因而会造成理解上的困难。

代码向量化通常只使用很少几个 Matlab 运算符和函数。这些运算符和函数通常用于完成索引操作或数组复制。我们可以简单地将这些运算符和函数分成 3 类，如下表所示（前两类是 Matlab 的基本内部操作，执行起来比较快；最后一类为优化的 M 文件代码，是用于完成数组复制的函数）：

运算符	描述
:	冒号操作符。例如，n:m 将创建一个以 n 开始，以 m 结束的行向量；n:inc:m 将创建一个以 n 开始，以 inc 为增量，以 m 或小于 m 结束的行向量。如果出现在数组索引（即括号）中，冒号（:）代表获取所有的元素。另外，如果 A(:)出现在等号右端，表明将 A 变为一个列向量；如果 A(:)出现在等号左端，表明使用等号右端的结果填充 A，而不需要重新为 A 分配内存
.'	非共轭转置操作符，使行和列互换
[ ]	方括号操作符，用于连接数组。比 cat 函数执行速度稍慢

内置函数	描述
all(x)	判断数组 x 的各元素是否都不为零。如果 x 中的所有元素都不为零，则返回 True
any(x)	判断数组 x 中是否有非零元素。如果 x 中有任意一个元素不是零，则返回 True
cat(Dim,A,B,…)	将 A,B,…沿 Dim 给出的那一维串联。执行速度比[ ]操作符快
cumsum(x)	求向量 x 的各元素的累积和
diff(x)	求 x 的各元素间的差分
end	末值索引，即代表给定那一维的最后一个元素
find(x) find(x,n) find(x,n,'last')	寻找 x 中非零元素的索引位置，通常比使用逻辑数组 x 直接进行数组寻址慢
logical(x)	将 x 转换成逻辑数据类型，使其能用于逻辑数组寻址
permute(A,Order)	广义转置。即重新排列 A 的各维，使其符合向量 Order 确定的顺序
prod(x)	求 x 中各元素的乘积
reshape(A,r,c)	将数组 A 变换成 r×c 的数组
sort(x) sort(x, 'descend')	将数组 x 按升序或降序排列
sum(x)	求 x 中各元素的和

M 文件函数	描述
ind2sub(Size,idx)	将 idx 中的单值索引转换成维数为 Size 的数组的下标值
ipermute(A,Order)	广义转置。是 permute(A,Order)的逆操作
kron(A,B)	求 A 和 B 的 Kronecker 张量积
meshgrid(x,y)	由向量 x 和 y 生成网格域
repmat(A,r,c)	通过复制数组 A 创建一个 r×c 的块状数组
shiftdim(A,n)	将数组 A 的各维循环平移 n 次
squeeze(x)	将数组 A 中的单独维删除
sub2ind(Size,r,c)	将维数为 Size 的数组的下标值 r 和 c 转换成单值索引

需要说明的是，如果上面的操作符、内置函数和 M 文件函数用于标量操作这一特殊情况也能明显提高运行的速度，则这一相应的数组操作过程也称为向量化。

38.2 JIT 加速

从 Matlab 6.5 开始，Matlab 解释器进行了改进，使得执行循环的处理时间变得最小。这些改进方法被统称为 JIT 加速器（JIT Accelerator）。这些改进在 Matlab 6.5 以后的各个版本中都存在。在 Matlab 7 中，JIT 加速器的性能优势已经涉及到了 Matlab 语法、数据类型、和数组大小。对于一段包含循环的 Matlab 代码，如果具有以下的特点或属性，就可以使用 JIT 加速进行性能优化：

（1）该循环语句为 For 循环语句。

（2）该循环仅包含下列数据类型：逻辑数组、字符串、双精度数据、低于 64 比特的整数数据类型。

（3）该循环的条件数组最高为三维数组。

（4）循环体内出现的变量均已在循环开始之前预先定义好。

（5）循环体内出现的变量均已预先分配好了内存，并且明确指定了内存大小和数据类型。

（6）循环体内的索引值都是循环变量，并且都是数量值，如，for i＝1:N，其中 i 为索引值。

（7）循环体内的函数都是 Matlab 的内置函数，不含有用户自定义的函数。

（8）如果循环体内包含条件语句（即后面将要讲到的 if-then-else 或 switch-case 结构），则该条件语句只包含标量比较操作。

（9）循环体内最多只包含一个赋值语句。

循环体内使用的数组维数越小，则 JIT 加速器对代码的优化越好。但随着数组维数的增加，无论是代码本身的计算时间还是对代码的运行管理时间都会增加，因此，JIT 加速器对整个执行时间的优化性能也会降低。

38.3 UP-DOWN 序列

作为第一个例子，我们来考虑下面的一个简单算法：令 N 为任意正整数，如果 N 是偶数，则将 N 除以 2，如果 N 是奇数，则将 N 乘以 3 再加 1，反复进行以上操作，直到 N 变成 1 为止。上述算法有一些很有趣的特性：首先，对于所有的 N，该算法都是收敛于 1 的；其次，对于有些 N，需要多次反复操作才能收敛到 1，而有些 N，例如 $N=2^m$，则可以迅速收敛到 1。我们可以分两个方面来研究上述算法：①研究对于给定的不同 N 值，该算法每一步计算所生成的值构成的序列。②研究对于给定的不同 N 值，该算法达到收敛所需要的迭代次数。下面我们首先从后一个研究开始。

首先，令 N 为标量，来编写基于单个数字的算法程序，该程序的 M 脚本文件如下：

```
% updown1.m
% up-down algorithm

N = 25;      % number to test
count = 0;  % iteration count
while N>1
    if rem(N,2)==0 % even, since division by 2 gives zero remainder
        N=N/2;
        count=count+1;
    else           % odd
        N=(3*N+1)/2;
        count=count+2;
    end
end
count  % display iteration count
```

上面的代码直接执行了 N=25 时算法所需的迭代次数。在代码中，有一点需要读者注意：当 N 为奇数时，N 乘以 3 加 1 所得的结果肯定是偶数，算法在下一步一定执行 N/2 这一步操作。因此，为了简化代码，程序将 N 乘以 3 加 1 与 N 除以 2 这两步结合在了一起，作为利用一个公式完成，但由于实际上它执行的是两步操作，所以计数值 count 在本步循环中每次都加 2。

下面，我们考虑当 N 为数组的情况。此时，我们需要找出对于 N 中的每一个元素，算法执行所需的迭代次数。实现这一任务的最直接的方法是使用 For 循环，其 M 脚本文件如下所示：

```
% updown2.m
% up-down algorithm

Nums = 25:50;  % numbers to test

for i=1:length(Nums)
    N=Nums(i);  % number to test
    count = 0;  % iteration count
        while N>1
```

```
        if rem(N,2)==0 % even
            N=N/2;
            count=count+1;
        else           % odd
            N=(3*N+1)/2;
            count=count+2;
        end
    end
    Counts(i)=count;
end
results=[Nums' Counts']
```

由上述代码可知，For 循环中使用了前面给出的基于标量的算法。在 For 循环体内，首先将向量 Nums 的第 i 个元素复制到 N 中；然后执行算法直到收敛；算法结束后将迭代计数值复制到 Counts 的第 i 个元素中。当 Nums 中所有的元素都计算完毕，最后显示计算的结果。这段代码违反了一个关键的内存分配原则，即变量 Counts 在每次进入 For 循环时都会被重新分配一个更大的内存空间。因此，上述代码很明显没有采用 Matlab 的 JIT 加速特性。

为了解决上述问题，Counts 必须按照下面代码中给出的方式进行内存分配：

```
% updown3.m
% up-down algorithm

Nums = 25:50;                % numbers to test
Counts=zeros(size(Nums));    % preallocate array
N = Nums(1);                 % predefine N data type and dimension
count = 0;                   % predefine count data type and dimension
for i=1:length(Nums)
    N=Nums(i); % number to test
    count = 0; % iteration count
        while N>1
        if rem(N,2)==0 % even
            N=N/2;
            count=count+1;
        else           % odd
            N=(3*N+1)/2;
            count=count+2;
        end
    end
    Counts(i)=count;
end
results=[Nums' Counts']
```

在上述代码中，每次 For 循环结束，都仅仅将当前的 count 插入到已经分配好内存的 Counts 中。注意：预先分配内存是向量化的第一个步骤，也是最重要的一个步骤。另外，由于 N 和 count 要在循环体内用到，因此对它们的数据类型和维数进行了预定义，这是该程序充分利用 Matlab 的 JIT 加速特性的一个重要步骤。

为了便于比较，下面将给出完成算法的向量化方案。在该向量化方案中，所有的输入数据都是被同时处理的，因此就不存在 For 循环。rem 函数将返回一个和与其输入相同大小的数组，这样在每次执行 While 循环时，所有的数字都被同时处理。下面是向量化方案的 M 脚本文件：

```
% updown4.m
% up-down algorithm

Nums = 25:50;                % numbers to test
N=Nums;                      % duplicate numbers
Counts=zeros(size(N));       % preallocate array

not1 = N>1;                  % True for numbers greater than one

while any(not1)

   odd = rem(N,2)~=0;     % True for odd values

   odd_not1 = odd&not1;  % True for odd values greater than one
   even_not1 = ~odd&not1; % True for even values greater than one

   N(even_not1) = N(even_not1)/2;  % Process evens
   Counts(even_not1)=Counts(even_not1)+1;

   N(odd_not1)=(3*N(odd_not1)+1)/2;         % Process odds
   Counts(odd_not1)=Counts(odd_not1)+2;

   not1 = N>1;      % Find remaining numbers not converged
end
results=[Nums' Counts']
```

当同时考虑到所有元素时，我们必须要找到一种方法可以不对那些已经收敛到 1 的数组元素再进行运算。程序中使用的 not1 = N>1;这条语句利用逻辑值指定了没有达到收敛的所有 N 值。在 While 循环中，未收敛的奇数和偶数值分别用逻辑变量 odd_not1 和 even_not1 指定。程序就是利用这两个逻辑变量处理 N 中相应的元素，并更新 Counts 的值的。

上面的矢量化方案使用了逻辑数组寻址。当然，我们可以使用 find 函数利用数值索引方式进行寻址。例如，奇数索引值可以通过 find(odd_not1)给出。不过，使用数值索引方式进行寻址会使代码使用的语句数增加，从而使代码执行速度变慢。

如果使用剖析器运行上面的 M 文件，用户就会发现大部分处理时间都被 odd = rem(N,2)~=0;这条语句占用了。这主要有两方面原因：第一个原因是，这条语句每次都寻找 N 中所有元素的余数，而不仅仅是剩余的未收敛的元素。例如，如果 N 中有 1000 个数据，并且只有 10 个数据是未收敛的，那么上述语句的求余计算中将有 990 个是不需要的。第二个原因是，求余函数会带来额外的开销，因为它毕竟是一个函数调用，并且内部含有错误检测。上述两个原因都充分说明，处理时间的占用都来自于对 rem 函数的调用。消除第一个原因的主要措施是将已收敛的数值从 N 中去除，不过这一步执行起来不那么容易。不过，我们可以通过用 rem 函数的定义代码代替该函数调用，来使额外开销最小化。

```
% updown5.m
% up-down algorithm

Nums = 25:50; % numbers to test
N = Nums;     % duplicate numbers
Counts = zeros(size(N)); % preallocate array

not1 = N>1;                    % True for numbers greater than one
while any(not1)

   odd = (N-2*fix(N/2))~=0;  % True for odd values

   odd_not1 = odd & not1;     % True for odd values greater than one
   even_not1 = ~odd & not1;   % True for even values greater than one

   N(even_not1) = N(even_not1)/2;         % Process evens
   Counts(even_not1) = Counts(even_not1)+1;

   N(odd_not1) = (3*N(odd_not1)+1)/2;      % Process odds
   Counts(odd_not1) = Counts(odd_not1)+2;

   not1 = N>1;                          % Find remaining numbers
end
results=[Nums' Counts']
```

如果使用剖析器再次运行上面的代码，会发现(N-2*fix(N/2))要比 rem(N,2)执行速度快。

最后，我们还可以通过使用整数算术运算来提高程序运行性能。由于 N 和 Counts 总是整数，因此，我们就可以将它们设置为整数数据类型。此时，我们也不需要使用(N-2*fix(N/2))来替换 rem 函数，而使用 odd = 2*(N/2)~=N 这条语句来判断是否为奇数。采用整数数据类型的程序代码如下所示：

```
% updown6.m
% up-down algorithm

Nums = 25:50;   % numbers to test
N = uint32(Nums);                  % duplicate numbers as uint32
Counts = zeros(size(N),'uint32'); % preallocate array as uint32

not1 = N>1;                    % True for numbers greater than one

while any(not1)

  odd = 2*(N/2)~=N;           % True for odd values

  odd_not1 = odd & not1;      % True for odd values greater than one
  even_not1 = ~odd & not1;     % True for even values greater than one

  N(even_not1) = N(even_not1)/2;         % Process evens
  Counts(even_not1) = Counts(even_not1)+1;

  N(odd_not1) = (3*N(odd_not1)+1)/2;      % Process odds
  Counts(odd_not1) = Counts(odd_not1)+2;

  not1=N>1;                 % Find remaining numbers
end
results=[Nums' Counts']
```

最后，我们来比较后四个函数的运行性能。首先，将每个函数的最后一条语句results=[Nums' Counts']去掉，并令Nums均为1:2049。然后使用剖析器观察每个函数的执行耗时，观察结果如下表所示：

函数	相对执行时间	备注
updown3	1.0	该函数为使用了Matlab的JIT加速特性的标量操作算法
updown5	1.2	该函数为不使用rem函数的向量化操作算法
updown4	1.5	该函数为使用rem函数的向量化操作算法
updown6	2.3	该函数为不使用rem函数的向量化操作算法，其中使用了uint32整数数据类型

上述结果充分显示了JIT加速器的巨大效能。惟一使用JIT加速器的函数updown3，要比其他三个函数的执行时间明显少许多。不过有一点使人惊奇，就是使用整数数据类型的函数updown6却是最慢的，本来它应该和updown5的执行时间相同。

需要指出的是，上面的结果只代表作者在自己的计算机上用Matlab 7验证的数据。并且剖析器的每一次运行所得到的结果是稍有不同的。如果读者要在自己的计算机上进行验证，可以从下面的网址上下载上面的示例代码：http://www.eece.maine.edu/mm。

38.4 范德蒙多矩阵

范德蒙多矩阵是许多线性代数问题都要用到的矩阵形式。对于一个n维向量x，其范德蒙多矩阵为：

$$V=\begin{bmatrix} x_1^m & x_1^{m-1} & \cdots & x_1 & 1 \\ x_2^m & x_2^{m-1} & \cdots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n^m & x_n^{m-1} & \cdots & x_n & 1 \end{bmatrix}$$

由上式可知，V的各列由x的每个元素的相同次幂组成。下面我们将讨论生成此矩阵的几种方法。

用户最先也最容易想到的一种方法是直接使用For循环来生成范德蒙多矩阵。具体代码如下面给出的M脚本文件所示：

```
% vander1.m
% construct a Vandermonde matrix.

x=(1:6)';         % column vector for input data
m=5;              % highest power to compute
V=[];

for i=1:m+1       % build V column by column
   V=[V x.^(m+1-i)];
end
```

上述方法中的矩阵 V 是由一个空矩阵逐列构建起来的。该方法存在许多缺点，最明显的一点就是每次循环都需要对 V 重新分配内存。因此，使上述代码向量化的第一步应该是为 V 预先分配内存，如下面给出的 M 文件所示：

```
% vander2.m
% construct a Vandermonde matrix.

x=(1:6)';          % column vector for input data
m=5;               % highest power to compute
n=length(x);       % number of elements in x
V=ones(n,m+1);     % preallocate memory for result

for i=0:m-1    % build V column by column
   V(:,i+1)=x.^(m-i);
end
```

上述方法中，V 在循环体外已被初始化为一个全 1 矩阵。在 For 循环体内只对 V 的各个列进行赋值。在 For 循环中，V 的最后一列未被赋值，这是因为最后一列的元素已经都是 1 了（V 已被初始化为一个全 1 矩阵），没有必要再进行 x.^0 的计算。不过，上面的代码依旧存在两个问题：首先，V 的各列是独立进行计算的，没有充分利用前一列的计算结果；其次，程序中仍然存在 For 循环。下面给出的 M 脚本文件可以解决第一个问题：

```
% vander3.m
% construct a Vandermonde matrix.

x=(1:6)';          % column vector for input data
m=5;               % highest power to compute
n=length(x);       % number of elements in x
V=ones(n,m+1);     % preallocate memory for result

for i=m:-1:1   % build V column by column
   V(:,i)=x.*V(:,i+1);
end
```

上面的代码改变了计算的方向，从矩阵 V 的倒数第二列元素开始逆向计算 V 的各个列，直到第一列为止。这样做的原因很简单，因为 V 的第 i 列等于第 i+1 列的元素乘以 x。上述实现方法即是 Matlab 函数 polyfit 和 vander 采用的方法。

不过，如果不消除 For 循环，我们仍很难进一步对算法进行优化。消除 For 循环不是一件简单的事，它需要用户具有灵活处理的能力，并且要对 Matlab 的函数非常熟悉。如果读者还记得本章第一节讲到的 3 类向量化操作符和操作函数，我们可以从那里找到解决方法，例如，其中的 repmat 和 cumprod 函数就可以用于对 vender3 进行改进。下面给出的 M 脚本文件就使用了 repmat 函数：

```
% vander4.m
% construct a Vandermonde matrix.

x=(1:6)';          % column vector for input data
```

```
m=5;            % highest power to compute
n=length(x);    % number of elements in x

p=m:-1:0;      % column powers
V=repmat(x,1,m+1).^repmat(p,n,1);
```

上例中两次使用了 repmat 函数，一次用于复制 x 以创建一个每列都由 x 构成的 m+1 列的矩阵；另一次则用于创建一个由应用到上述 m+1 列矩阵的幂组成的矩阵。利用这两个矩阵，用户就可以通过逐元素指数运算生成希望的结果。上述方法与 vander2.m 一样，对矩阵的每一列单独计算，没有使用其他列的信息。为了能够使用其他列的信息，我们可以使用 cumprod 函数重新编写上述代码，结果如下面给出的 M 脚本文件所示：

```
% vander5.m
% construct a Vandermonde matrix.

x=(1:6)';           % column vector for input data
m=5;                % highest power to compute
n=length(x);        % number of elements in x

V=ones(n,m+1);    % preallocate memory for result
V(:,2:end)=cumprod(repmat(x,1,m),2);
V=V(:,m+1:-1:1);  % reverse column order
```

上面的代码在使用了 repmat 函数复制 x 后，使用了 cumprod 函数来计算 V 的列。因为 cumprod 是从左向右执行的，所以最后的结果需要将 V 的各列反向。上述方法还是用到了一个函数调用——repmat 函数。我们可以使用数组寻址来替换此函数，以加速程序运行的速度。下面给出了使用数组寻址的 M 脚本文件：

```
% vander6.m
% construct a Vandermonde matrix.

x=(1:6)';      % column vector for input data
m=5;          % highest power to compute
n=length(x);    % number of elements in x

V=ones(n,m+1); % preallocate memory for result
V(:,2:end)=cumprod(x(:,ones(1,m)),2); % avoid call to repmat
V=V(:,m+1:-1:1); % reverse column order
```

在上述六个实现方法中，vander3 和 vander6 是最快的。vander3 使用的方法也是 Matlab 函数 polyfit 和 vender 采用的方法，该方法比 vander6 更容易阅读，并且需要的内存也少。另外几个方法无论在内存分配还是计算量上，都不如 vander3 和 vander6 有效。

38.5 重复值的创建和计数

本节，我们来讨论下面的问题：假设有一个包含数据的向量 x，和一个等长的包含非负整数的向量 n，构建一个向量，使得 x(i)被重复 n(i)次后放在该向量中。例如，如果给定

x=[3 2 0 5 6]，n=[2 0 3 1 2]，则生成的向量应为 y=[3 3 0 0 0 5 6 6]。请注意，因为 n(2)=0，所以 x(2)未在结果 y 中出现。

除了创建重复值外，它的逆问题：确定并计算向量中重复的值也是用户经常关心的问题。即，给定向量 y，找出向量 x 和 n。

我们首先来考虑创建重复值的问题。最简单最直接的方法是标量方法，如下面给出的 M 脚本文件所示：

```
% repeat1.m
% repeated value creation and counting

x = [3 2 0 5 6];  % data to repeat
n = [2 0 3 1 2];  % repeat counts

y=[];
for i=1:length(x)
    y=[y repmat(x(i),1,n(i))];
end
```

上述方法通过使用方括号将所有的重复值依次串联起来得到了结果向量。该方法不能使用 JIT 加速特性，因为在循环体内出现了 rempat 函数，并且每次迭代都需要为 y 重新分配内存。我们可以通过为 y 预分配内存和采用索引的方法解决这一问题，如下面的 M 脚本文件所示：

```
% repeat2.m
% repeated value creation and counting

x = [3 2 0 5 6];      % data to repeat
n = [2 0 3 1 2];      % repeat counts

nz = n==0;            % locations of zero elements
n(nz) = [];           % eliminate zero counts
x(nz) = [];           % eliminate corresponding data
y = zeros(1,sum(n));  % preallocate output array

idx = 1;                       % pointer into y
for i=1:length(x)
   y(idx:idx+n(i)-1) = x(i);  % fill y using scalar expansion
   idx = idx+n(i);             % next pointer location
end
```

在上面的代码中，在 x 和 n 进入循环体之前去掉了计数值（即 n(i)的值）为 0 的元素，因为这些元素对结果不产生影响，但却耗费计算时间。另外，上面的方法还使用 sum(n)确定 y 的长度，并使用 idx 变量确定下一个数据应被放在 y 的什么地方。

现在，我们还需要考虑使用向量化解决上述的重复值问题。为了实现该方法，我们首先需要明白 x 与 y 到底存在什么样的关系。就上述 M 文件中的 x 和 n 而言，当去除了计数值为 0 的元素以后，x、n 和 y 将分别为：

```
>> x
x =
     3     0     5     6
>> n
n =
     2     3     1     2
>> y
y =
     3     3     0     0     0     5     6     6
```

如果我们能够创建一个索引向量 idx=[1 1 2 2 2 3 4 4]，那么 x 和 y 的关系为：

```
>> idx =[1 1 2 2 2 3 4 4];
>> y = x(idx)
y =
     3     3     0     0     0     5     6     6
```

因此，如果我们能够生成一个索引向量，那么只需要使用 y=x(idx)一条语句就可以得到 y。这样我们就不用总想着如何将 x 的元素放置到 y 中正确的位置了。可见，索引是实现向量化的最常见方法，它往往比数据本身更为重要。

我们也很容易找到 n 和 idx 之间的关系。另外，由于 idx 看起来很像一个累积和，因此，我们可以使用 cumsum 函数来创建 idx。具体方法是：我们可以先生成一个只含有 0 和 1 的数组，并且该数组只在 idx 的元素值发生变化的位置为 1，其余的位置为 0，然后再使用累加和的方法计算 idx，如下面的代码所示：

```
>> tmp = [1 0 1 0 0 1 1 0]
tmp =
     1     0     1     0     0     1     1     0
>> idx = cumsum(tmp)
idx =
     1     1     2     2     2     3     4     4
```

我们也很容易发现，tmp 中的非零值与 n 有关。为了找出它们之间的关系，我们先来看数组 n 的累加和：

```
>> csn = cumsum(n)
csn =
     2     5     6     8
```

如果丢掉 csn 中的最后一个值，并将剩余的值都加 1，就得到了 tmp 中 1 的索引位置（不过该索引位置不包括第一个 1 的位置，因为第一个 1 的位置永远为 1）。这样，我们就可以把 1 的所有索引位置求出，如下面的代码所示：

```
>> tmp2 = [1 csn(1:end-1)+1]
tmp2 =
     1     3     6     7
```

从结果可以看出，tmp2 中的值确实指定了 tmp 中所有的 1 的位置。现在，我们就可以利用 tmp2 创建 tmp，并通过 tmp 来求 y，代码如下：

```
>> tmp = zeros(1,csn(end))      % preallocate with all zeros
tmp =
    0      0      0      0      0      0      0      0
>> tmp(tmp2) = 1                % poke in ones with scalar expansion
tmp =
    1      0      1      0      0      1      1      0
>> idx = cumsum(tmp)            % form the desired cumsum
idx =
    1      1      2      2      2      3      4      4
>> y = x(idx)                   % idx does the rest!
y =
    3      3      0      0      0      5      6      6
```

将上面的方法结合在一起，我们就可以得到创建重复值的向量化方案，如下面的 M 脚本文件所示：

```
% repeat3.m
% repeated value creation and counting

x = [3 2 0 5 6];                 % data to repeat
n = [2 0 3 1 2];                 % repeat counts

nz = n==0;                       % locations of zero elements
n(nz) = [];                      % eliminate zero counts
x(nz) = [];                      % eliminate corresponding data
csn = cumsum(n);                 % cumulative sum of counts
tmp = zeros(1,csn(end));         % preallocate memory
tmp([1 csn(1:end-1)+1]) = 1;     % poke in ones
idx = cumsum(tmp);               % index vector
y = x(idx);                      % let array indexing do the work
```

我们也使用了剖析器来分析上述 3 个 M 脚本文件的耗时特性，在验证过程中，使用的 x 和 n 比文件内部的数组大得多。下表给出了验证的结果：

函数	相对执行时间	备注
repeat3	1.0	向量化方案
repeat2	4.0	使用 JIT 加速特性的标量操作
repeat1	55	不含变量内存预分配的非优化解决方案

从上面的结果可以看出，向量化方案比 JIT 加速方案执行速度快。这与我们上一节在范德蒙多矩阵中的测试结果截然相反。不过，矢量化方案也存在一些不利因素，那就是代码不是十分清晰。如果不亲自对其进行运行并观察中间变量的结果，用户恐怕很难说出每条向量化后的语句是如何工作的。

下面我们来考虑重复值问题的逆问题，即重复值的定位和计数。也就是说，给定 y，求 x 和 n。当然，解决这一问题的最直接最简单的方法仍是采用 For 循环的非向量化方法，如下面的 M 脚本文件所示：

```
% repeat4.m
% repeated value creation and counting
% inverse operation

y = [3 3 0 0 0 5 6 6];      % data to examine

x = y(1);                   % beginning data
n = 1;                      % beginning count
idx = 1;                    % index value
for i=2:length(y)
   if y(i)==x(idx)          % value matches current x

      n(idx) = n(idx)+1;    % increment current count

   else                     % new value found

      idx = idx+1;          % increment index
      x(idx) = y(i);        % poke in new x
      n(idx) = 1;           % start new count
   end
end
```

上面的代码使用了一个简单的If-Else-End结构判断y中的一个元素是否是当前重复出现的值。如果是，计数值便加1，如果不是，则创建一个新的重复值。如果读者认真阅读的话，可以发现每次Else部分语句执行时，都会对x和n重新分配内存。因此，这种方法无法使用JIT加速特性进行性能优化。

对上面的代码进行优化的第一个步骤是使用内存预分配，如下面的M脚本文件所示：

```
% repeat5.m
% repeated value creation and counting
% inverse operation

y = [3 3 0 0 0 5 6 6];      % data to examine

x = zeros(size(y));         % preallocate results
n = zeros(size(y));

x(1) = y(1);                % beginning data
n(1) = 1;                   % beginning count
idx = 1;                    % index value

for i=2:length(y)
   if y(i)==x(idx)          % value matches current x

      n(idx) = n(idx)+1;    % increment current count

   else % new value found
      idx = idx+1;          % increment index
      x(idx) = y(i);        % poke in new x
      n(idx) = 1;           % start new count
   end
end
```

```
nz = (n==0);                     % find elements not used
x(nz) = [];                      % delete excess allocations
n(nz) = [];
```

由于 x 和 n 的长度都是未知的（不过都小于 y 的长度），因此，在对它们进行内存预分配时，将它们的长度都设为 y 的长度。而在计算结束后，再将多余的内存删除。

不过，上面的代码中仍然含有 For 循环。要去掉 For 循环，实现矢量化，我们需要对 y 进行一些研究。我们仍以前面的 y 为例，首先查看 y 中的元素，如下面的代码所示：

```
>> y
y =
     3     3     0     0     0     5     6     6
```

下面我们对该向量使用 diff 函数，如下面的代码所示：

```
>> diff(y)
ans =
     0    -3     0     0     5     1     0
```

如果将 ans 向量向右移动一个元素，那么它的非零元素的位置就是 y 中新的重复值出现的位置。此外，y 的第一个元素肯定是一个新的重复值。因此，对移位后的 ans 向量使用逻辑运算就可以得到重复值出现的位置，如下面的代码所示：

```
>> y
y =
     3     3     0     0     0     5     6     6
>> tmp = [1 diff(y)]~=0
tmp =
     1     0     1     0     0     1     1     0
```

根据逻辑变量 tmp，就可以通过对 y 进行逻辑寻址得到向量 x：

```
>> x = y(tmp)
x =
     3     0     5     6
```

得到 x 以后，我们还需要求 x 中的每一个元素的重复次数 n。我们不难发现，重复次数等于 tmp 中相邻两个 1 之间的距离。因此，我们需要得到 tmp 中 1 的索引，并求出这些索引之间的差值，如下面的代码所示：

```
>> find(tmp)
ans =
     1     3     6     7
>> diff(ans)
ans =
     2     3     1
```

上述索引差值给出了 x 中除最后一个元素外的所有元素的重复次数，这是由于我们调用 diff 函数造成的。我们可以在 tmp 的末尾添加一个非 0 元素来解决这一问题，如下面的代码所示：

```
>> find([tmp 1])
ans =
    1       3       6       7       9
>> n = diff(ans)
n =
    2       3       1       2
```

对上述步骤进行总结，我们可以得到下面的代码，用于实现 x 和 n 的求解：

```
% repeat6.m
% repeated value creation and counting
% inverse operation

y=[3 3 0 0 0 5 6 6]; % data to examine

tmp = [1 diff(y)]~=0;
x = y(tmp);
n = diff(find([tmp  1]));
```

repeat6 给出了实现矢量化的一个更加紧凑的方案，它只用了 3 行代码就实现了向量化，而 repeat5 却用了 17 行。当然，repeat6 也是最难阅读的向量化代码（向量化代码本身都不容易阅读），如果用户不亲自执行并观察执行结果的话，这 3 行代码恐怕很难理解。

我们用剖析器比较了上述 3 个代码(repeat4、repeat5、repeat6)，发现 repeat4 最慢，repeat5 和 repeat6 大致相等。如果重复值给出的比较少（即 y 的长度相对较短），则不在 repeat4 中进行内存预分配也不会造成多大的速度减慢。不过，如果 y 的长度比 x 的长度大很多，则 repeat5 中的后 3 行删除程序确实是十分耗时的。

需要指出的是，如果 y 中含有 Inf 和 NaN 这样的元素，则上述实现方案都是无法使用的。因为对 diff 函数来说，Inf 之间的差值为 NaN，而 NaN 之间的差值也为 NaN。

从下面给出的 mmrepeat 函数（该函数中封装了上面讲到的一些算法）中，我们可以看到创建和计数重复值的重要作用：

```
function [y,m]=mmrepeat(x,n)
%MMREPEAT Repeat or Count Repeated Values in a Vector.
% MMREPEAT(X,N) returns a vector formed from X where X(i) is repeated
% N(i) times. If N is a scalar it is applied to all elements of X.
% N must contain nonnegative integers. N must be a scalar or have the same
% length as X.
%
% For example, MMREPEAT([1 2 3 4],[2 3 1 0]) returns the vector
% [1 1  2 2 2  3]    (extra spaces added for clarity)
%
% [X,N]=MMREPEAT(Y) counts the consecutive repeated values in Y returning
% the values in X and the counts in N. Y=MMREPEAT(X,N) and [X,N]=MMREPEAT(Y)
% are inverses of each other if N contains no zeros and X contains unique
% elements.

if nargin==2 % MMREPEAT(X,N) MMREPEAT(X,N)  MMREPEAT(X,N)  MMREPEAT(X,N)
   xlen=length(x);
```

```
    nlen=length(n);
    if ndims(x)~=2 | numel(x)~=xlen
      error('X Must be a Vector.')
    else
      [r,c]=size(x);
    end
    if any(n<0) | any(fix(n)~=n)
      error('N Must Contain NonNegative Integers.')
    end
    if ndims(n)~=2 | numel(n)~=nlen | (nlen>1 & nlen~=xlen)
      error('N Must be a Scalar or Vector the Same Size as X.')
    end
    x=reshape(x,1,xlen); % make x a row vector

    if nlen==1 % scalar n case, repeat all elements the same amount
      if n==0 % quick exit for special case
        y=[];
        return
      end
      y=x(ones(1,n),:); % duplicate x to make n rows each containing x
      y=y(:);           % stack each column into a single column
      if r==1           % input was a row so return a row
        y=y.';
      end
    else % vector n case
      iz=find(n>0);           % take out elements to be repeated zero times
      x=x(iz);
      n=n(iz);
      csn=cumsum(n);
      y=zeros(1,csn(end));    % preallocate temp/output variable
      y(csn(1:end-1)+1)=1;    % mark indices where values increment
      y(1)=1;                 % poke in first index
      y=x(cumsum(y));         % use cumsum to set indices
      if c==1                 % input was a column so return a column
        y=y.';
      end
    end
elseif nargin==1 % MMREPEAT(Y)  MMREPEAT(Y)  MMREPEAT(Y)  MMREPEAT(Y)
    xlen=length(x);
    if ndims(x)~=2 | prod(size(x))~=xlen
      error('Y Must be a Vector.')
    else
      [r,c]=size(x);
    end
    x=reshape(x,1,xlen); % make x a row vector
    xnan=isnan(x);
    xinf=isinf(x);
    if any(xnan|xinf) % handle case with exceptions
      ntmp=sum(rand(1,4))*sqrt(realmax);    % replacement for nan's
      itmp=1/ntmp;                          % replacement for inf's
```

```
      x(xnan)=ntmp;
      x(xinf)=itmp.*sign(x(xinf));
      y=[1 diff(x)]~=0;           % places where distinct values begin
      m=diff([find(y) xlen+1]);   % counts
      x(xnan)=nan;                % poke nan's and inf's back in
      x(xinf)=inf*x(xinf);

    else % x contains only algebraic numbers
      y=[1 diff(x)]~=0;           % places where distinct values begin
      m=diff([find(y) xlen+1]);   % counts
    end
    y=x(y); % the unique values
    if c==1
      y=y.';
    end
else
    error('Incorrect Number of Input Arguments.')
end
```

38.6 差分求和

本节将讨论数组元素之间差分和的计算。例如，如果x=[3 1 2 6 3 1 –1]，那么x的差分和是y=[4 3 8 9 4 0]。差分和可以看作是Matlab的diff函数的对偶函数，或补函数。与diff函数一样，我们也希望差分和函数也可以处理输入的任一维，而无需考虑输入的维数。在将差分和函数推广到n维之前，我们先来考虑一下该函数用于向量和矩阵的情况。用于向量的情况很简单，如下例所示：

```
>> x = [ 3 1 2 6 3 1 -1]
x =
     3     1     2     6     3     1    -1
>> x(1:end-1)
ans =
     3     1     2     6     3     1
>> x(2:end)
ans =
     1     2     6     3     1    -1
>> y = x(1:end-1)+x(2:end)
y =
     4     3     8     9     4     0
```

上面的例子中仅仅利用了对数组的简单寻址就实现了差分和计算，不过这种方法只适用于x是一个行向量或列向量的情况。如果x是一个矩阵，那么我们定义默认的操作是沿着列的方向进行差分和计算，如下例所示：

```
>>x = magic(4) % 2-D data
x =
    16     2     3    13
```

```
     5      11      10       8
     9       7       6      12
     4      14      15       1
>> y = x(1:end-1,:)+x(2:end,:)
y =
    21      13      13      21
    14      18      16      20
    13      21      21      13
```

我们也可以指定沿着行的方向进行操作，如下例所示：

```
>> y = x(:,1:end-1)+x(:,2:end)
y =
    18       5      16
    16      21      18
    16      13      18
    18      29      16
```

我们发现，这两个结果的行数和列数正好是相反的。那么，如果我们在操作前先对 x 进行转置，操作后再对 x 进行转置，那么就可以通过一种方法执行这两个方向的计算。例如，我们可以通过上面的途径再次计算沿行方向的差分和操作，代码如下所示：

```
>> tmp = x';    % transpose data
>> y = tmp(1:end-1,:)+tmp(2:end,:);
>> y = y'       % transpose result
y =
    18       5      16
    16      21      18
    16      13      18
    18      29      16
```

上述操作也可以通过 n 维函数 permute 和 ipermute 来完成，这两个函数是矩阵转置运算符的广义扩展。例如，下面给出了采用这两个函数计算沿行方向的差分和的代码：

```
>> tmp = permute(x,[2 1])
tmp =
    16       5       9       4
     2      11       7      14
     3      10       6      15
    13       8      12       1
>> y = tmp(1:end-1,:)+tmp(2:end,:);
>> y = ipermute(y,[2 1])
y =
    18       5      16
    16      21      18
    16      13      18
    18      29      16
```

下面，我们来考虑三维数据的差分和，如下例所示：

```
>> x = cat(3,hankel([3 1 6 -1]),pascal(4))
x(:,:,1) =
```

```
     3        1        6       -1
     1        6       -1        0
     6       -1        0        0
    -1        0        0        0
x(:,:,2) =
     1        1        1        1
     1        2        3        4
     1        3        6       10
     1        4       10       20
>> y = x(1:end-1,:,:)+x(2:end,:,:) % diff sum along row dimension
y(:,:,1) =
     4        7        5       -1
     7        5       -1        0
     5       -1        0        0
y(:,:,2) =
     2        3        4        5
     2        5        9       14
     2        7       16       30
```

读者可以发现，这里使用的操作同前面的基本相同，只不过多使用了一个冒号表示 x 的第三维（即页维）。如果用户将 1:end-1 和 2:end 这两条语句移动到其他维的位置，那么就可以针对其他维求差分和，例如：

```
>> y = x(:,:,1:end-1)+x(:,:,2:end)
y =
     4        2        7        0
     2        8        2        4
     7        2        6       10
     0        4       10       20
```

该例子求出了 x 的页维之间的差分和。

上面给出了将差分和算法推广到 n 维的一种方法。其中，我们在对等号右侧的 x 索引时用到了逗号分隔列表。如果我们能够创建一个包含该索引列表的单元数组，就可以使用逗号分隔列表语法来计算差分和，如下例所示：

```
>> y = x(1:end-1,:,:)+x(2:end,:,:) % duplicate this case
y(:,:,1) =
     4        7        5       -1
     7        5       -1        0
     5       -1        0        0
y(:,:,2) =
     2        3        4        5
     2        5        9       14
     2        7       16       30

>> c1 = {(1:3) ':' ':'}      % first set of indices
c1 =
    [1×3 double]        ':'      ':'
>> c2 = {(2:4) ':' ':'}      % second set of indices
c2 =
    [1×3 double]        ':'      ':'

>> y = x(c1{:})+x(c2{:})     % use comma separated list syntax
```

```
y(:,:,1) =
    4        7        5       -1
    7        5       -1        0
    5       -1        0        0
y(:,:,2) =
    2        3        4        5
    2        5        9       14
    2        7       16       30
```

上例充分展示了逗号分隔列表语法的强大功能。注意，在这里不能使用关键字 end，因为 end 只在直接作为变量的索引使用时才有意义。因此，上例的 c1 和 c2 中均包含实际的数字索引。

下面给出的 M 脚本文件即是使用逗号分隔列表语法计算差分和的完整代码：

```
% diffsum1.m
% compute differential sum along a given dimension

x = cat(3,hankel([3 1 6 -1]),pascal(4)) % data to test
dim = 1  % dimension to work along

xsiz = size(x)
xdim = ndims(x)

tmp = repmat({':'},1,xdim) % cells of ':'
c1 = tmp;
c1{dim} = 1:xsiz(dim)-1    % poke in 1:end-1
c2 = tmp;
c2(dim) = {2:xsiz(dim)}    % poke in 2:end

y = x(c1{:})+x(c2{:})     % comma separated list syntax
```

在 Matlab 中执行 diffsum1（不要带分号），会生成如下的输出：

```
x(:,:,1) =
     3        1        6       -1
     1        6       -1        0
     6       -1        0        0
    -1        0        0        0
x(:,:,2) =
     1        1        1        1
     1        2        3        4
     1        3        6       10
     1        4       10       20
dim =
     1
xsiz =
     4        4        2
xdim =
     3
tmp =
     ':'      ':'      ':'
c1 =
     [1×3 double]       ':'      ':'
```

```
c2 =
   [1×3 double]        ':'      ':'
y(:,:,1) =
   4         7         5        -1
   7         5        -1         0
   5        -1         0         0
y(:,:,2) =
   2         3         4         5
   2         5         9        14
   2         7        16        30
```

其中，x 是输入数据，dim 指定了要计算哪一维的差分和。repmat 将根据 x 的大小和维数信息，生成一个单元数组用于寻址 x 中的所有元素。然后将该单元数组复制到 c1 和 c2 中，并利用 dim 参数修改 c1 和 c2。最后，使用逗号分隔列表语法生成最终结果。

上面的方法并非计算多维数组的差分和的惟一方法。我们在前面的介绍中讲到，使用 permute 和 ipermute 函数可以对 x 进行转置，以便于计算沿行方向的差分和。这一方法也可以用在三维数组的差分和算法中，如下面的 M 脚本文件所示：

```
% diffsum2.m
% compute differential sum along a given dimension

x = cat(3,hankel([3 1 6 -1]),pascal(4)) % data to test
dim = 3                             % dimension to work along

xsiz = size(x);
n = xsiz(dim);                      % size along desired dim
xdim = ndims(x);                    % # of dimensions

perm = [dim:xdim 1:dim-1]           % put dim first
x = permute(x,perm)                 % permute so dim is row dimension
x = reshape(x,n,[])                 % reshape into a 2D array

y = x(1:n-1,:)+x(2:n,:)             % Differential sum along row dimension

xsiz(dim) = n-1                     % new size of dim dimension
y = reshape(y,xsiz(perm))           % put result back in original form
y = ipermute(y,perm)                % inverse permute dimensions
```

其中，perm 是对 x 进行转置操作的图样向量（即决定如何对 x 进行转置的一个向量）。转置后，x 将被重新构造成一个二维数组。如果 x 是三维数组，我们需要将 x 的各页转置为一页，并作为 x 的附加列进行排列，然后对 x 进行差分和计算，最后，将 x 反向置换还原为原始的形状。但有一点需要注意，计算差分和后的 x 会在相应维的方向上长度减 1。

在 Matlab 中执行 diffsum2（不要带分号），会生成如下的输出：

```
x(:,:,1) =
      3      1      6     -1
      1      6     -1      0
      6     -1      0      0
     -1      0      0      0
x(:,:,2) =
      1      1      1      1
      1      2      3      4
```

```
     1     3     6    10
     1     4    10    20

dim =
     3
perm =
     3     1     2
x(:,:,1) =
     3     1     6    -1
     1     1     1     1
x(:,:,2) =
     1     6    -1     0
     1     2     3     4
x(;,:,3) =
     6    -1     0     0
     1     3     6    10
x(:,:,4) =
    -1     0     0     0
     1     4    10    20
x =
  Columns 1 through 12
     3     1     6    -1     1     6    -1     0     6    -1     0     0
     1     1     1     1     1     2     3     4     1     3     6    10
  Columns 13 through 16
    -1     0     0     0
     1     4    10    20
 y =
   Columns 1 through 12
     4     2     7     0     2     8     2     4     7     2     6    10
   Columns 13 through 16
     0     4    10    20
 xsiz =
     4     4     1
 y(:,:,1) =
     4     2     7     0
 y(:,:,2) =
     2     8     2     4
 y(:,:,3) =
     7     2     6    10
 y(:,:,4) =
     0     4    10    20
 y =
     4     2     7     0
     2     8     2     4
     7     2     6    10
     0     4    10    20
```

就我们上面讲到的两种扩展函数的方法而言，diffsum2 所采用的多维情况更可能出现在 Matlab 的 M 文件中。不过就速度而言，这两种方法没有什么差别。

38.7 结构体处理

结构体是 Matlab 中一种很方便的数据结构。利用结构体，用户可以将多个相关的数据

组合到一个单独变量中，并使用描述性的域名来标识它们。另外，用户也可以很方便地将一组变量放入一个结构体中，并在任何需要的时候取出这些变量。为了演示结构体的用法，我们首先来看一下如何将一组变量放入结构体中，并使结构体的各个域名与变量原来的名字相同。在命令窗口中完成这一任务很简单，用户只需将同名的变量分配给字段即可，如下例所示：

```
>> a = eye(2) % test data
a =
     1     0
     0     1
>> b = 'String'
b =
String
>> c = cell(2)
c =
    []     []
    []     []
>> y.a = a; % store variables in a structure
>> y.b = b;
>> y.c = c
y =
    a: [2×2 double]
    b: 'String'
    c: {2×2 cell}
```

上述过程的逆过程也很简单。比如，仍以上面的 y 为例，我们可以将 y 的各域赋给一个变量，如下面的代码所示：

```
>> a =y.a
a =
     1     0
     0     1
>> b = y.b
b =
String
>> c = y.c
c =
    []     []
    []     []
```

上述将变量组合到结构体，和将结构体分解到变量的操作，可以用下面的 M 函数文件来完成：

```
function varargout=mmv2struct(varargin)
%MMV2STRUCT Pack/Unpack Variables to/from a Scalar Structure.
% MMV2STRUCT(X,Y,Z,...) returns a structure having fields X,Y,Z,...
% containing the corresponding data stored in X,Y,Z,...
% Inputs that are not variables are stored in fields named ansN
% where N is an integer identifying the Nth unnamed input.
%
% MMV2STRUCT(S)assigns the contents of the fields of the scalar structure
```

```
% S to variables in the calling workspace having names equal to the
% corresponding field names.
%
% [A,B,C,...]=MMV2STRUCT(S) assigns the contents of the fields of the
% scalar structure S to the variables A,B,C,... rather than overwriting
% variables in the caller. If there are fewer output variables than
% there are fields in S, the remaining fields are not extracted. Variables
% are assigned in the order given by fieldnames(S).

if nargin==0
   error('Input Arguments Required.')

elseif nargin==1 % Unpack Unpack Unpack Unpack Unpack Unpack Unpack Unpack
   arg=varargin{1};
   if ~isstruct(argin)||length(argin)~=1
      error('Single Input Must be a Scalar Structure.')
   end
   names=fieldnames(argin);
   if nargout==0 % assign in caller
      for i=1:length(names)
         assignin('caller',names{i},argin.(names{i}));
      end
   else          % deal fields into variables in caller
      varargout=cell(1,nargout);        % preallocate output
      for i=1:nargout
         varargout{i}=argin.(names{i});
      end
   end

else % Pack Pack Pack Pack Pack Pack Pack Pack Pack Pack Pack Pack Pack Pack
   args=cell(2,nargin);
   num=1;
   for i=1:nargin % build cells for call to struct
      args(:,i)={inputname(i);varargin{i}};
      if isempty(args{1,i})
         args{1,i}=sprintf('ans%d',num);
         num=num+1;
      end
   end
   varargout{1}=struct(args{:}); % create struct using comma-separated list
end
```

下面我们来考虑如何将结构体分解到变量中。如果用户在调用 mmv2struct(structvar)时没有输出参数，mmv2struct 函数将使用 For 循环反复调用函数 assignin 为工作区中名为 mmv2struct 的变量赋值。在每次 For 循环中，都会创建与结构体域名相匹配的变量。语句段 arg.(names{i})用于查找保存在 names{i}中的结构体 argin（等于 structvar）的域名。然后将 arg.(names{i})的内容赋给工作区的变量 names{i}。

那么函数又是如何将变量组合到结构体中的呢？首先，函数创建一个单元数组，用于包含 struct 函数所需的参数（例如，struct('field1',values1,'field2',values2,...)）。为了叙述方便，假设 struct 的参数为'field1'、'field2'等，它们是 args 变量的第一行，以及 values1、values2

等，它们为 args 变量的第二行。那么，语句 struct(args{:})中的 args{:}就会使用逗号分隔列表语法创建 struct 函数所需的一行参数。

38.8 反向插值

在本书第 19 章，我们使用函数 interp1(x,y,xi)讨论了一维插值。尽管该函数提供了很多有用特性，但它有一个前提条件，就是对每一个 xi，都只有一个 y 与之对应。如果该条件不满足，则函数将立即停止并报错。

有时候，我们还需要进行反方向的插值。即，给定 y 值，寻找满足 $y=f(x)$的所有 x 值。为了更好地对反向插值进行描述，我们先来看一个简单的例子，如下面的代码和图形所示：

```
>> x = (1:10).';    % sample data
>> y = cos(pi*x);

>> yo = -0.2;   % inverse interpolation point
>> xol = [x(1);x(end)];
>> yol = [yo;yo];

>> plot(x,y,xol,yol)
>> xlabel X
>> ylabel Y
>> set(gca,'Ytick',[-1 yo 0 1])
>> title('Figure 38.1:Inverse Interpolation')
```

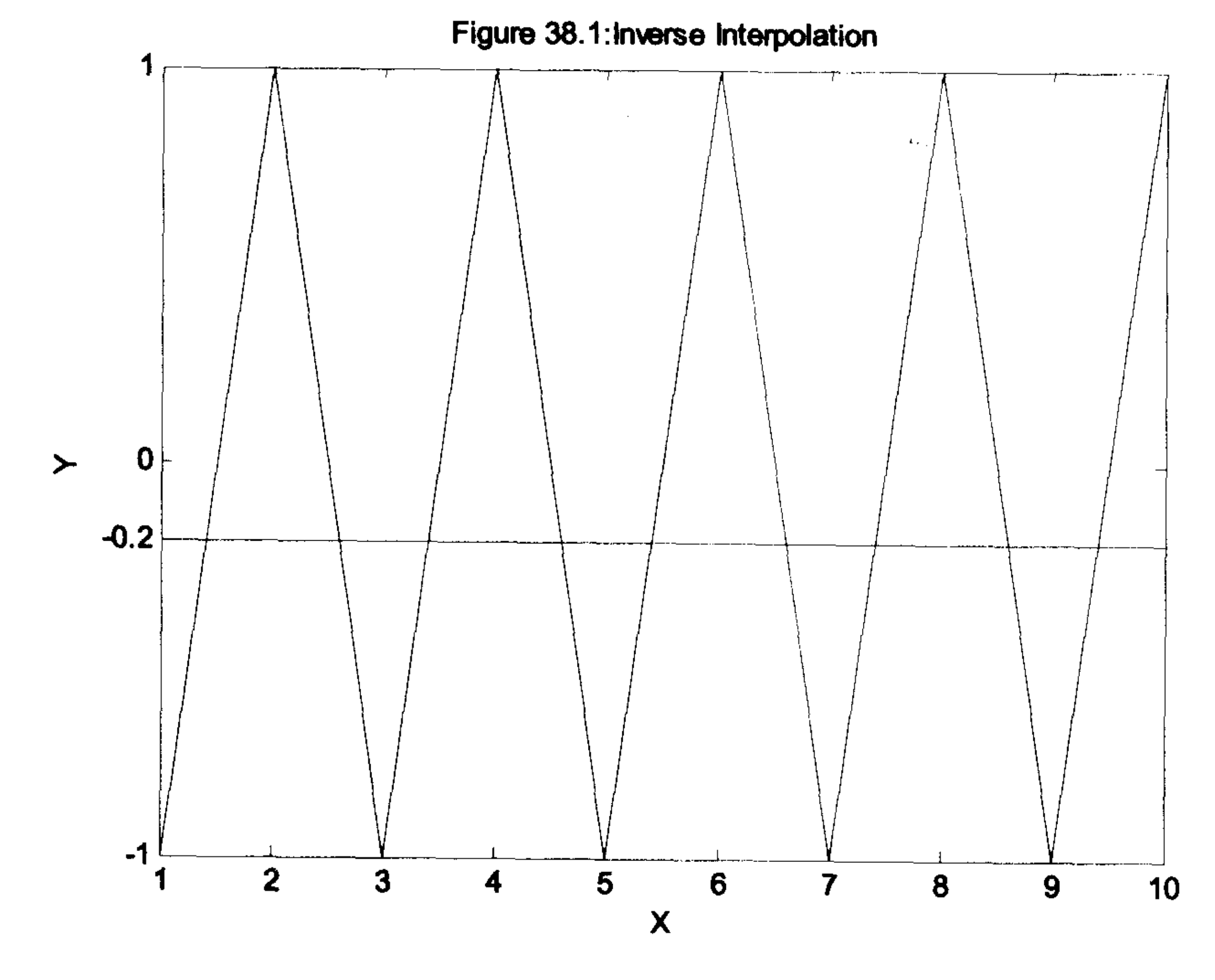


图 38.1 反向插值

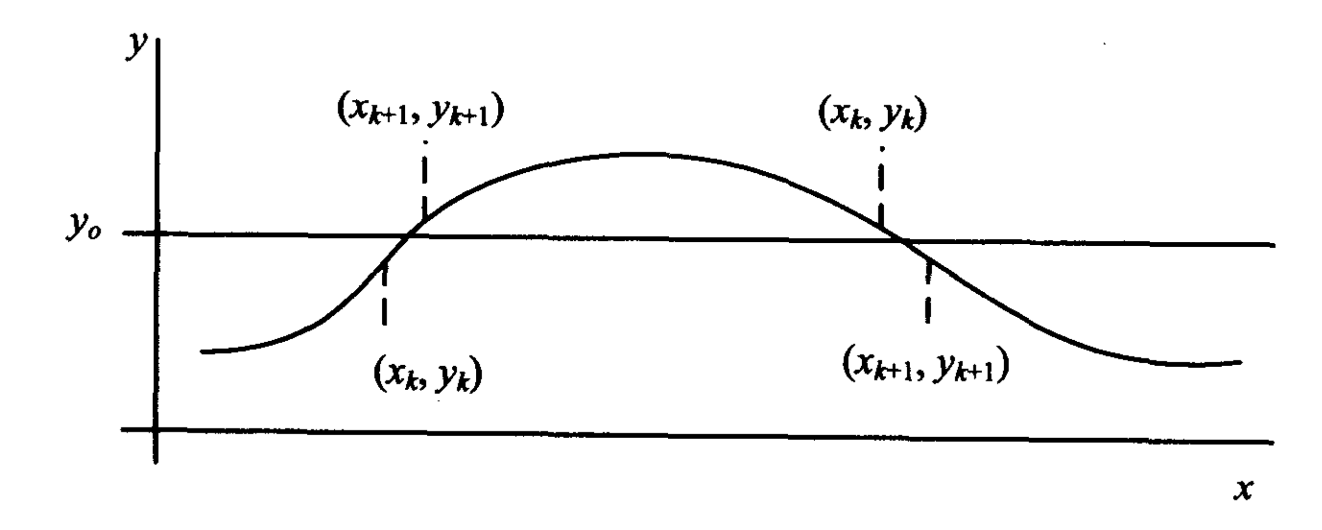
对于图 38.1 中的三角函数，使用 interp1 来根据给定的 x 值寻找 y 值是很容易的事，因为，对于每一个 x 值，只有一个 y 值与之对应。但是，反向插值就不那么简单了，因为，

对于一个[-1, 1]范围内的 y 值，会有多个 x 值与之对应。例如，对于图 38.1 中的 yo = −0.2 这条水平线而言，就同时存在 9 个 x 值与一个 y（等于−0.2）对应。很显然，我们是没有办法用 interp1 函数来寻找这些 x 值的，如下面的代码所示：

```
>> interp1(y,x,yo)
??? Error using ==> interp1
The data abscissae should be distinct.
```

造成上述错误的原因在于 interp1 函数需要的单调条件没有得到满足。下面我们就需要创建一个 M 文件函数来完成上述操作。

为了解决这一问题，我们必须先找出图 38.1 中交错排列的所有数据点。如下面给出的图形所示，在一个小的区域（该区域内的曲线近似为单调），如果一个数据点出现在插值点 y_o 的下方，则下一个数据点一定出现在插值点的上方，反之亦然。当我们找到$(x_k,y_k)(x_{k+1},y_{k+1})$后，就可以使用线性插值寻找给定 y_o 时的 x_o 值。



下面的代码段给出了实现上述过程的一个简单的非向量化的方案：

```
% invinterp1

x = (1:10).';              % sample data
y = cos(pi*x);

yo =-0.2;                  % chosen inverse interpolation point

if yo<min(y) || yo>max(y)        % quick exit if no values exist
   xo = [];
else                             % search for the desired points

   n = length(y);
   xo = zeros(size(y))+nan;      % preallocate space for found points
   alpha = 0;

   for k=1:n-1  % look through all data pairs
      if ( y(k)<yo && y(k+1)>=yo ) ||...      % below then above
         ( y(k)>yo && y(k+1)<=yo )            % above then below

         alpha=(yo-y(k))/(y(k+1)-y(k)); % distance between x(k+1) and x(k)
         xo(k)=alpha*(x(k+1)-x(k))+x(k); % linearly interpolate using alpha
      end
```

```
    end
    xo = xo(~isnan(xo));          % get rid of unneeded preallocated space
    yo = repmat(yo,size(xo));     % duplicate yo to match xo points found
end
```

注意，上述代码用一个 NaNs 数组为插值数据点 xo 预分配了内存。这里没有使用全 0 和全 1 数据分配内存是因为 0 和 1 可能是有效的数据点值。从某种意义上讲，这种预分配可能有些多余，因为通常情况下 xo 的长度比输入数据的长度小得多。代码的倒数第二行语句将没有用到的 xo 数据点删除，最后一条语句将 yo 复制成与 xo 等长的数组。将 yo 复制成与 xo 等长是为了作图方便，如 plot(xo,yo,'o')。

很明显，上述代码是可以进行向量化的，这是因为我们在其中可以进行基于数组的逻辑比较操作。为了验证向量化的可行性，我们来看下面的示例代码：

```
>> x = (1:10);    % sample data
>> y = cos(pi*x);
>> yo = -0.2;   % chosen inverse interpolation point
>> below = y<yo    % True where below yo
below =
    1   0   1   0   1   0   1   0   1   0
>> above = y>=yo   % True where at or above yo
above =
    0   1   0   1   0   1   0   1   0   1
```

根据上例中 x 和 y 的选择，数据点总是一个在 yo 之上，一个在 yo 之下，呈间隔排列。也就是说，yo 将被包含在 below(k)与 above(k+1)为真和 above(k)与 below(k+1)为真的两个连续的数据点之间。如果我们将 below 和 above 数组均右移一位，则可以得到下面的一些测试结果：

```
>> n =length(y);
>> below(1:n-1) & above(2:n)        % below(k) and above(k+1)
ans =
    1   0   1   0   1   0   1   0   1

>> above(1:n-1) & below(2:n)        % above(k) and below(k+1)
ans =
    1   0   1   0   1   0   1   0   1
```

将上述两个数组结合，可以得到所有的第 k 个数据点。紧接在这些数据点之后的是第(k+1)个数据点，它们由下面的代码求出：

```
>> kth=(below(1:n-1)&above(2:n))|(above(1:n-1)&below(2:n)); % True at k points
>> kp1=[false kth];                                  % True at k+1 points
```

上述两个逻辑数组就指定了需要插值的 y 的数据点。下面的代码给出了使用上面介绍的插值算法实现反向插值的向量化解决方案：

```
% invinterp2

x = (1:10).'; % sample data
```

```
y = cos(pi*x);
yo = -0.2;  % chosen inverse interpolation point

n = length(y);

if yo<min(y) || yo>max(y) % quick exit if no values exist
   xo = [];
else                       % find the desired points

   below = y<yo;  % True where below yo
   above = y>=yo; % True where at or above yo

   kth = (below(1:n-1)&above(2:n))|(above(1:n-1)&below(2:n)); % point k
   kp1 = [false; kth];                                    % point k+1

   alpha = (yo - y(kth))./(y(kp1)-y(kth)); % distance between x(k+1) and x(k)
   xo = alpha.*(x(kp1)-x(kth)) + x(kth); % linearly interpolate using alpha

   yo = repmat(yo,size(xo)); % duplicate yo to match xo points found
end
```

上面的代码使用逻辑比较和逻辑处理代替了 For 循环。另外，也没有使用 find 函数，因为该函数通常需要额外的计算，会造成实现速度的降低。上面整个程序都是使用数组运算进行线性插值。其中，由于 xo 的大小在程序运行过程中是可变的，因此没有必要对其进行内存预分配。上述向量化方案的一个不足点是创建了太多的大型中间变量。

我们利用剖析器对前面介绍的两个实现方案进行了分析。结果发现这两个方案具有几乎相同的性能。有时候，JIT 加速方案要好一些，而有时候向量化方案要好一些。由于这两个方案不分伯仲，下面我们就给出一个采用向量化方案的 M 函数文件（至于采用 JIT 加速方案的 M 函数文件，请读者自己完成）：

```
function [xo,yo]=mminvinterp(x,y,yo)
%MMINVINTERP 1-D Inverse Interpolation.
% [Xo, Yo]=MMINVINTERP(X,Y,Yo) linearly interpolates the vector Y to find
% the scalar value Yo and returns all corresponding values Xo interpolated
% from the X vector. Xo is empty if no crossings are found. For
% convenience, the output Yo is simply the scalar input Yo replicated so
% that size(Xo)=size(Yo).
% If Y maps uniquely into X, use INTERP1(Y,X,Yo) instead.
%
% See also INTERP1.

if nargin~=3
   error('Three Input Arguments Required.')
end
n = numel(y);
if ~isequal(n,numel(x))
   error('X and Y Must have the Same Number of Elements.')
end
if ~isscalar(yo)
   error('Yo Must be a Scalar.')
```

```
end

x=x(:); % stretch input vectors into column vectors
y=y(:);

if yo<min(y) || yo>max(y) % quick exit if no values exist
   xo = [];
   yo = [];
else                      % find the desired points

   below = y<yo;          % True where below yo
   above = y>=yo;         % True where at or above yo

   kth = (below(1:n-1)&above(2:n))|(above(1:n-1)&below(2:n)); % point k
   kp1 = [false; kth];                                         % point k+1

   alpha = (yo - y(kth))./(y(kp1)-y(kth));
                                 % distance between x(k+1) and x(k)
   xo = alpha.*(x(kp1)-x(kth)) + x(kth);
                                 % linearly interpolate using alpha

   yo = repmat(yo,size(xo)); % duplicate yo to match xo points found
end
```

上述函数有一个重要的应用：寻找两个曲线的交点。例如，现在有两条曲线，由下列代码创建：

```
>> x = linspace(0,10);
>> y = sin(x);
>> z = 2*cos(2*x);
>> plot(x,y,x,z)
>> xlabel X
>> yLabel Y
>> title 'Figure 38.2: Intersection of Two Curves'
```

这两条曲线的交点由它们的差值的过零点组成。根据上面给出的曲线，使用前面给出的 mminvinterp 函数，这些交点可以用下面的代码求出：

```
>> xo = mminvinterp(x,y-z,0);  % find zero crossing of difference
>> yo = interp1(x,y,xo);       % find corresponding y values
>> plot(x,y,x,z,xo,yo,'o')  % regenerate plot showing intersection points
>> xlabel X
>> ylabel Y
>> title 'Figure 38.3: Intersection Points'
```

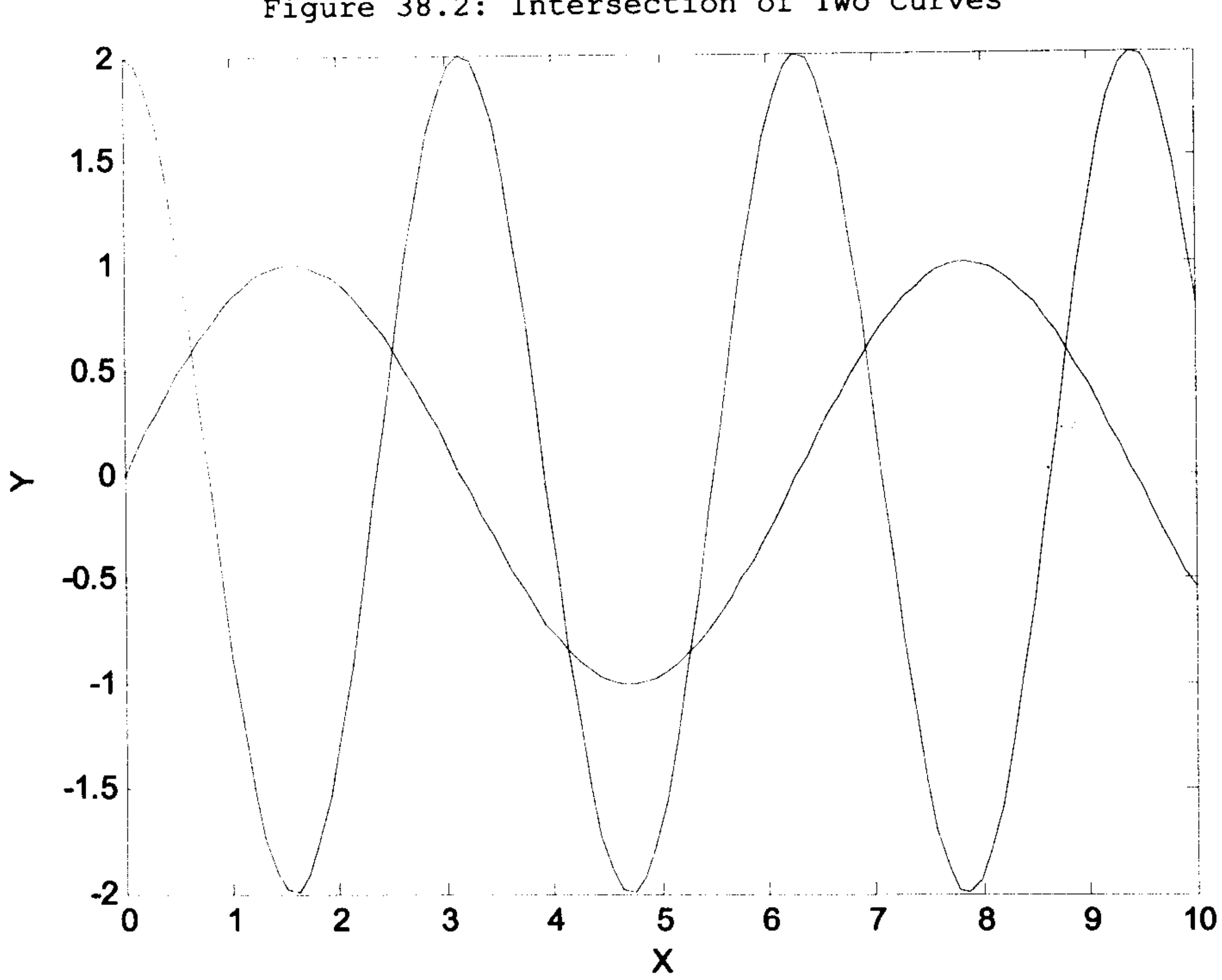


图 38.2　两条曲线相交

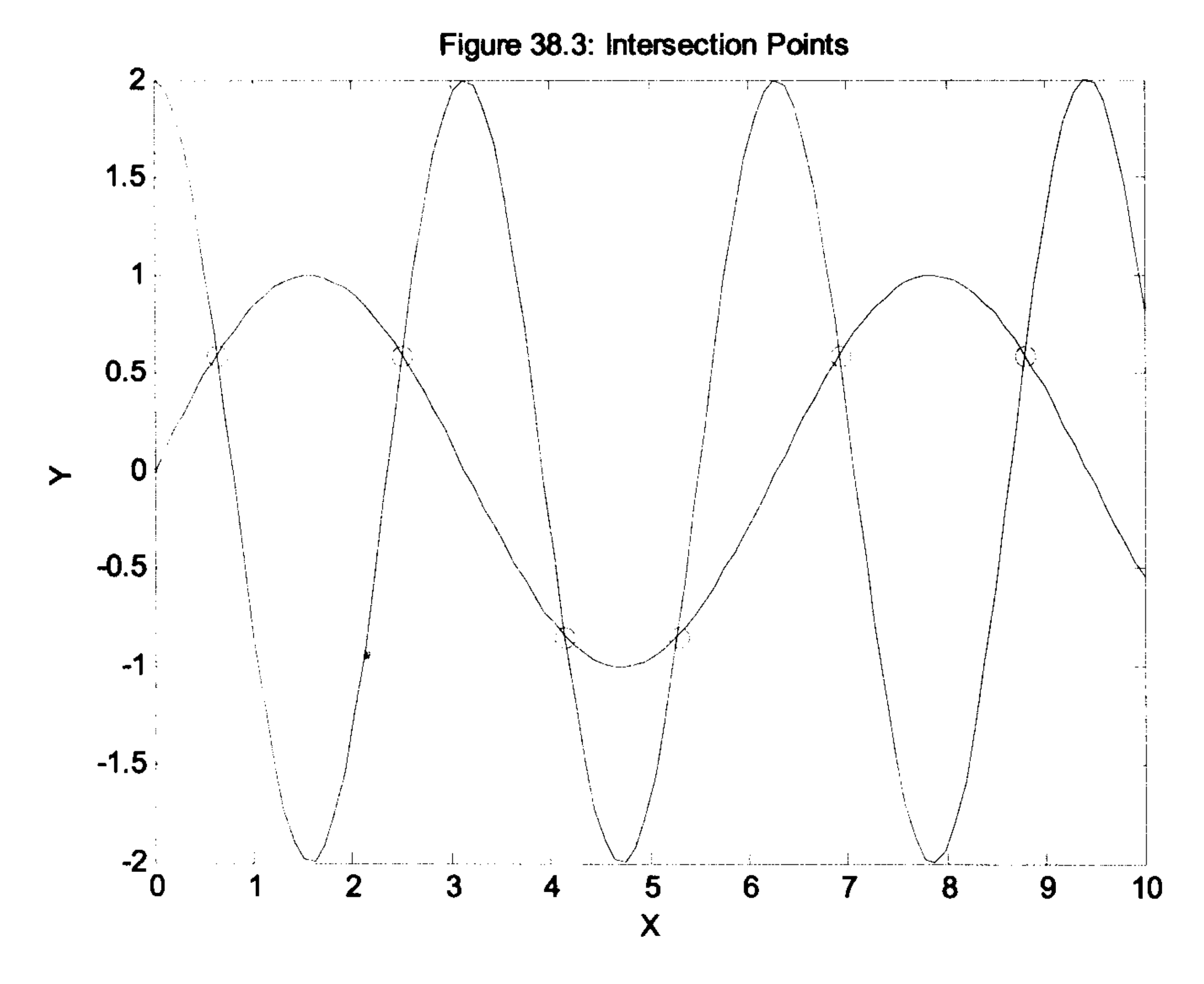


图 38.3　两条曲线的交点

38.9 多项式曲线拟合

我们在第 20 章已经讲到，Matlab 中执行多项式曲线拟合的函数是 polyfit。多项式曲线拟合是一项最基本的数值分析问题，我们有必要对其进行更深入的阐述。首先，一个多项式的一般表达方式为：

$$y = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}$$

由上式可以看出，一个 n 阶多项式具有 $n+1$ 个系数。为了计算方便，我们规定，随着 x 阶数逐渐降低，与之对应的系数的下标将逐渐升高（如 x^n 的系数为 p_1，x^{n-1} 的系数为 p_2）。这样的话，上述多项式可以用下面的矩阵形式描述：

$$y = [x^n \quad x^{n-1} \quad \cdots \quad x \quad 1] \cdot \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \\ p_{n+1} \end{bmatrix}$$

在上述描述形式中，多项式的系数已被表示成了一个列向量。

我们通常讲的曲线拟合是指，在给定一组数据 $\{x_i, y_i\}$（其中 $i=1,2,\cdots,N$）的条件下，利用上面介绍的多项式的矩阵描述形式计算多项式的系数的过程。将给定的数据代入到上面的关系式中，并写成矩阵形式，则为：

$$\begin{bmatrix} x_1^n & x_1^{n-1} & \cdots & x_1 & 1 \\ x_2^n & x_2^{n-1} & \cdots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_N^n & x_N^{n-1} & \cdots & x_N & 1 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \\ p_{n+1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

我们发现，上式最左边的矩阵其实是一个范德蒙多矩阵（这一类型的矩阵我们在本章 38.4 节讨论过）。如果 $N = n+1$，则范德蒙多矩阵就是一个方阵。另外，如果各数据点（即 $x_1, x_2, \cdots, x_N$）互不相同，则该矩阵又是一个满秩矩阵，这时可以使用 Matlab 中的左除运算符（\）求出惟一的一个多项式向量 $p = [p_1 \quad p_2 \quad \cdots \quad p_n \quad p_{n+1}]'$，即 p=V\y，其中，V 是范德蒙多矩阵，y 是前面公式中右侧的向量。

另外，当 $N \geq n+1$ 并且数据点互不相同时，范德蒙多矩阵的行数将多于列数，此时上面的共识不存在确定的解。这种情况下，Matlab 的左除运算符（即 p=V\y）计算出来的将是使均方误差最小的多项式系数向量。例如，下面的代码在不使用 polyfit 函数的情况下，再一次求解了图 20.2 给出的示例：

```
% polyfit1.m
% find polynomial coefficients without polyfit
```

```
x = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1]'; % column vector data
y = [-.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.30 11.2]';
n = 2;            % desired polynomial order

pm = polyfit(x,y,n) % MATLAB polyfit result

% create Vandermonde matrix using code from vander3.m
m = length(x);   % number of elements in x
V = ones(m,n+1); % preallocate memory for result
for i=n:-1:1     % build V column by column
   V(:,i) = x.*V(:,i+1);
end

p = (V\y);  % find least squares solution
p = p' % convert to row vector to match MATLAB's convention for polynomials
```

运行上面的代码，可以发现使用 polyfit 函数和不使用 polyfit 函数所得到的结果是一样的，如下所示：

```
>> polyfit1
pm =
    -9.8108     20.129      -0.031671
p =
    -9.8108     20.129      -0.031671
```

在上面的代码中，我们还有必要考虑潜在的数值问题。尤其是范德蒙多矩阵的元素是从 1 到 x^n 变化的，当 x 点距离 1 很远时，上述程序可能就会面临精度问题。例如，如果将上述程序中的多项式阶数增加到 4 阶，x 向量扩大 10^4 倍，则范德蒙多矩阵可由下面的代码求出：

```
>> x = 1e4*[0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1]'; % scale x data by 1e4
>> n = 4;                 % change order to 4
>> m = length(x);         % number of elements in x
>> V = ones(m,n+1);       % preallocate memory for result
>> for i=n:-1:1           % build V column by column
    V(:,i) = x.*V(:,i+1);
end
>> V
V =
                0           0           0       0     1
           1e+012      1e+009      1e+006    1000     1
         1.6e+013      8e+009      4e+006    2000     1
         8.1e+013    2.7e+010      9e+006    3000     1
        2.56e+014    6.4e+010    1.6e+007    4000     1
        6.25e+014   1.25e+011    2.5e+007    5000     1
       1.296e+015   2.16e+011    3.6e+007    6000     1
       2.401e+015   3.43e+011    4.9e+007    7000     1
       4.096e+015   5.12e+011    6.4e+007    8000     1
       6.561e+015   7.29e+011    8.1e+007    9000     1
           1e+016      1e+012      1e+008   10000     1
```

由上面的结果可以看出，V 的取值范围为从 1 到 10^{16}。这种巨大的取值差距将会给调用 polyfit 函数求多项式系数造成一些麻烦，如下面的代码所示：

```
>> p = polyfit(x,y,n)
Warning: Rank deficient, rank = 4, tol = 3.1633e+001.
(Type "warning off MATLAB: RankDeficientMatrix" to suppress this warning.)
p =
    3.0675e-015    -4.8147e-011    9.5904e-008    0.0018927       0
```

由于我们已经达到了双精度类型所能进行的数学运算极限，因此要想解决这一问题，我们就必须将 x 的数据值按比例缩小，以便使范德蒙多矩阵的各元素间的差距不那么大。在众多的数值比例缩放方法中，均值和标准偏差方法是比较好的一种。也就是说，此时我们不再构建以 x 为自变量的多项式，而是构建以 z 为自变量的多项式，其中 z 的定义如下：

$$z=\frac{x-x_m}{s}$$

其中，x_m 和 s 分别为 x 的均值和标准偏差。将数据点减去均值，相当于将坐标原点平移到均值处，除以标准偏差意味着减小数据点之间的差距。仍以前面的四阶多项式为例，如果以 z 为自变量，则范德蒙多矩阵可以采用下面的代码求出：

```
>> xm = mean(x)
>> s = std(x)
>> z = (x - xm)/s;
>> m = length(z);  % number of elements in x
>> V = ones(m,n+1);  % preallocate memory for result
>> for i=n:-1:1     % build V column by column
    V(:,i) = z.*V(:,i+1);
end
>> V
V =
       5.1653     -3.4263      2.2727    -1.5076   1
       2.1157     -1.7542      1.4545     -1.206   1
      0.66942    -0.74007     0.81818   -0.90453   1
      0.13223    -0.21928     0.36364   -0.60302   1
    0.0082645    -0.02741    0.090909   -0.30151   1
            0           0           0          0   1
    0.0082645     0.02741    0.090909    0.30151   1
      0.13223     0.21928     0.36364    0.60302   1
      0.66942     0.74007     0.81818    0.90453   1
       2.1157      1.7542      1.4545      1.206   1
       5.1653      3.4263      2.2727     1.5076   1
```

这时，范德蒙多矩阵的数据取值集中了许多，这样我们就可以毫不费力地进行多项式曲线拟合了。根据上述数据，我们可以求得多项式系数为：

```
>> p = (V/y)'
p =
    0.26689      0.58649      -1.6858      2.4732       7.7391
```

与缩放前求得的系数相比，上述这些系数更符合我们的需要。

读者需要注意的是，上述系数实际上求得的是下面的多项式：

$$y = p_1 z^n + p_2 z^{n-1} + \cdots + p_n z + p_{n+1}$$

也就是说，现在多项式是 z 的函数。从上面的分析我们可以得出，根据给定的数据 x，求拟合多项式的系数的步骤主要有两步：首先，使用 x 的均值 x_m 和标准偏差 s 将 x 转化为 z，转化公式为：$z = \frac{x - x_m}{s}$；然后，使用拟合函数求多项式的系数。下面的代码给出了这两步的具体实现方法：

```
>> xi = 1e4*[0.25  0.35  0.45];      % sample data for polynomial evaluation
>> zi = (xi-xm)/s;                   % convert to z data
>> yi = polyval(p,zi)                % evaluate using polyval
yi =
    4.752        6.2326       7.326
```

如果用户不想手动进行自变量转化，则可以使用 polyfit 和 polyval 这两个函数，这时用户只需在函数调用时添加两个附加变量就可以了。例如，前面的几行代码如果使用这两个函数完成，我们可以采用下面的方法：

```
>> [p,Es,mu] = polyfit(x,y,n);
>> p
p =
    0.26689      0.58649     -1.6858      2.4732       7.7391
>> yi = polyval(p,xi,Es,mu)
yi =
    4.752        6.2326       7.326
```

其中，polyfit 函数的可选输出参数 mu 包含的是原始数据的均值和标准偏差。将此变量提供给 polyval 函数，可以使该函数在计算多项式之前先进行自变量转化。从执行结果来看，手动转化和函数转化的效果是一样的。

上面代码中的 Es 是一个结构体变量，它用于处理计算过程出现的误差。有关这一参数的详细介绍和用法请读者参考 Matlab 的联机帮助文档。

除了数据缩放外，我们还可能要考虑的问题是加权多项式曲线拟合。也就是说，有时原始数据中的某些数据点要比其他数据点具有更大的重要性。如果我们事先知道这一事实，就需要在进行曲线拟合时考虑到这一点。这样，我们最终求得的多项式就需要反映出原始数据点的权值。

Matlab 函数 polyfit 是无法直接执行加权多项式曲线拟合的，因此，我们就需要设计自己的函数来完成该任务。有多种方法可供我们选用，其中最简单的方法就是在计算范德蒙多矩阵之前添加加权值。例如，假如我们需要用一个三阶多项式拟合 5 个数据点，并且，第三个数据点的信任度（即加权值）是其他数据点的 α 倍，这样，我们需要求解的矩阵方程就变为：

$$\begin{bmatrix} x_1^3 & x_1^2 & x_1 & 1 \\ x_2^3 & x_2^2 & x_2 & 1 \\ \alpha \cdot x_3^3 & \alpha \cdot x_3^2 & \alpha \cdot x_3 & \alpha \\ x_4^3 & x_4^2 & x_4 & 1 \\ x_5^3 & x_5^2 & x_5 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \alpha \cdot y_3 \\ y_4 \\ y_5 \end{bmatrix}$$

由于矩阵的第三行乘上了一个系数α，因此该矩阵方程在求解时的误差也将受加权因子α的影响。因此，我们在计算最小均方误差时尤其需要迫使第三个数据点的误差小于其他的数据点。也就是说，α越大，第三个数据点处的误差就需要越小。

一般情况下，矩阵的每一行都可以设置加权因子，而不仅仅是第三行。例如，下面给出的 M 脚本文件就设置了两个加权因子（分别是第四行和第七行）：

```
% polyfit2.m
% find weighted polynomial coefficients

x = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1]';
y = [-.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.30 11.2]';
n = 3;
pm = polyfit(x,y,n)          % MATLAB polyfit result

% create Vandermonde matrix using code from vander3.m

m = length(x);               % number of elements in x
V = ones(m,n+1);             % preallocate memory for result
for i=n:-1:1                 % build V column by column
  V(:,i) = x.*V(:,i+1);
end

w = ones(size(x));           % default weights of one
w(4) = 2;                    % weight 4th point by 2
w(7) = 10;                   % weight 7th point by 10

V = V.*repmat(w,1,n+1);      % multiply rows by weights
y = y.*w;                    % multiply y by weights

p = (V\y)'                   % find polynomial
```

运行上述代码，并与前面的例子比较，我们发现加权和不加权的多项式是不同的，如下面的结果所示：

```
>> polyfit2
pm =
   16.076      -33.924      29.325       -0.6104
p =
   28.576      -50.761      34.104       -0.67441
```

除了上面的方法外，我们还可以使用 Matlab 函数 lscov 来求解。该函数主要用于求矩阵方程的加权最小二乘解。我们仍以上面的数据为例，如果使用 lscov 求解，其完整的 M 脚本文件如下所示：

```
% polyfit3.m
% find weighted polynomial coefficients using lscov

x = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1]';
y = [-.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.30 11.2]';
n = 3;
pm = polyfit(x,y,n) % MATLAB polyfit result

% create Vandermonde matrix using code from vander3.m

m = length(x);   % number of elements in x
V = ones(m,n+1); % preallocate memory for result
for i=n:-1:1     % build V column by column
   V(:,i) = x.*V(:,i+1);
end

w = ones(size(x)); % default weights of one
w(4) = 2^2;    % here weights are the square of those used in polyfit2
w(7) = 10^2;

p = lscov(V,y,w)'
```

上述程序的执行结果如下：

```
>> polyfit3
pm =
     16.076      -33.924      29.325      -0.6104
p =
     28.576      -50.761      34.104      -0.67441
```

为了比较前面介绍的 3 种方法（polyfit1、polyfit2、polyfit3）的性能，我们可以采用下面的 M 脚本文件将这些方法的执行结果在同一个图形中显示出来：

```
% mm3804.m
% find weighted polynomial coefficients using lscov

x = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1]';
y = [-.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.30 11.2]';
n = 3;
pm = polyfit(x,y,n); % MATLAB polyfit result

% create Vandermonde matrix using code from vander3.m

m = length(x);   % number of elements in x
V = ones(m,n+1); % preallocate memory for result

for i=n:-1:1     % build V column by column
   V(:,i) = x.*V(:,i+1);
end

w = ones(size(x)); % default weights of one
w(4) = 2^2;
w(7) = 10^2;

p = lscov(V,y,w)';
```

```
xi = linspace(0,1,100);
ym = polyval(pm,xi);
yw = polyval(p,xi);

pt = false(size(x));  % logical array pointing to weighted points
pt([4 7]) = true;

plot(x(~pt),y(~pt),'x',x(pt),y(pt),'o',xi,ym,'--',xi,yw)
xlabel('x'), ylabel('y=f(x)')
title('Figure 38.4: Weighted Curve Fitting')
```

运行上面的程序，可以得到下面的图形：

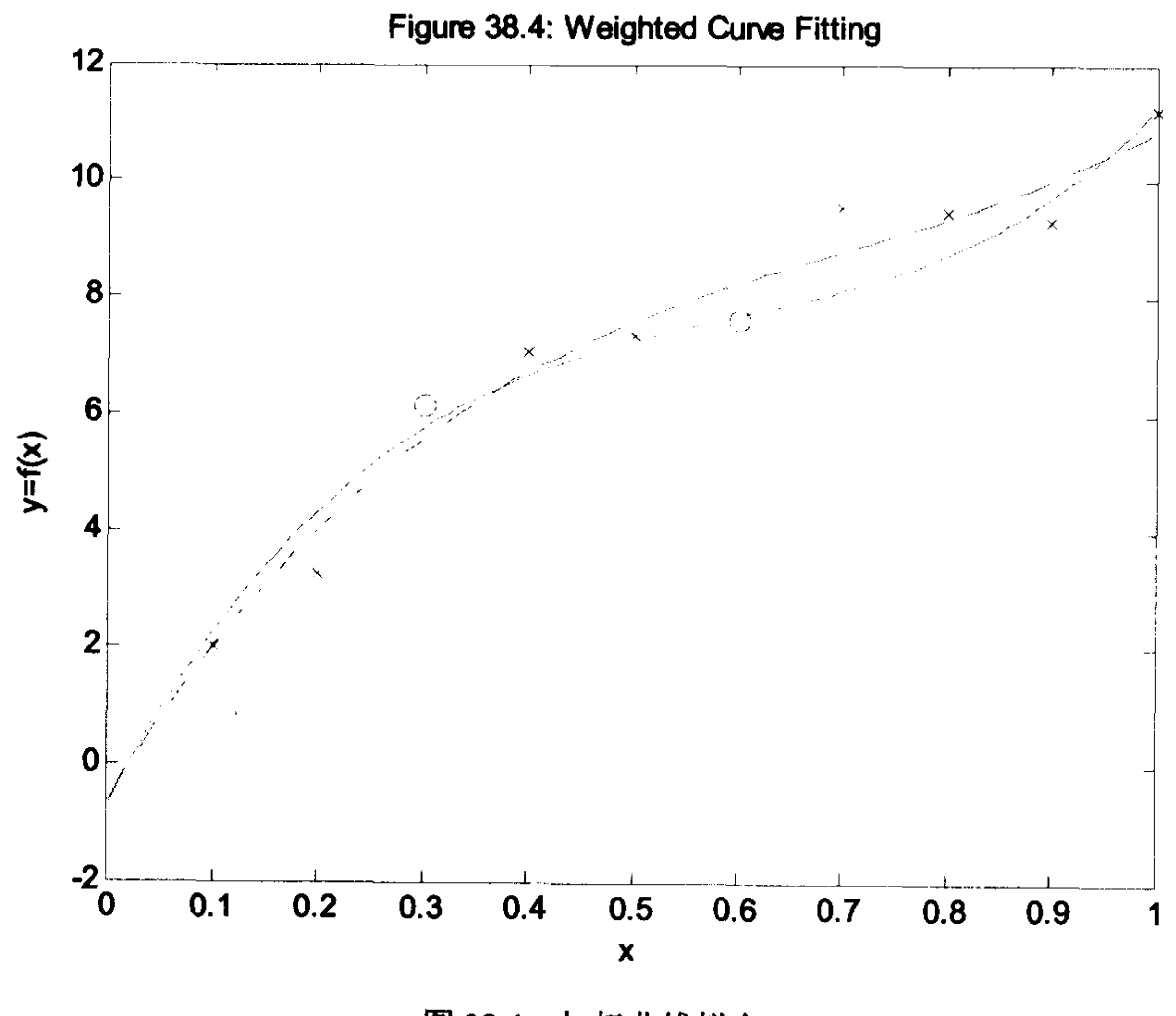


图 38.4　加权曲线拟合

上图中，加权数据用'o'表示，不加权的数据用'x'表示。很明显，加权多项式拟合曲线要比非加权多项式拟合曲线更接近两个加权点。另外，由于第七个点的加权因子较大，因此，加权曲线也更加接近该点。

38.10　非线性曲线拟合

前面讲到的多项式曲线拟合之所以很通用，是因为它可以用一个矩阵方程表示，并可以采用线性的最小二乘技术求解。对用户而言，该方法最重要最方便的地方或许在于它不需要初始假设，因为这样用户就不需要在一个 n 维解空间寻找最优解了。不过，通常情况下，未知的系数并不是呈线性变化的，这样，我们就必须从一个初始假设开始，在一个解空间中进行搜索。例如，如果我们要用下面的函数拟合数据：

$$f(t)=a+be^{\alpha t}+ce^{\beta t}$$

当α和β为已知常数，a、b、c为需要寻找的未知向量，则上述函数可以表示为：

$$f(t)=[1 \quad e^{\alpha t} \quad e^{\beta t}]\cdot\begin{bmatrix}a\\b\\c\end{bmatrix}$$

将一组已知的数据$\{t_i, y_i=f(t_i)\}, i=1,2,\cdots,N$代入上式，并写成矩阵形式，可得：

$$\begin{bmatrix}1 & e^{\alpha t_1} & e^{\beta t_1}\\1 & e^{\alpha t_2} & e^{\beta t_2}\\ \vdots & \vdots & \vdots\\1 & e^{\alpha t_N} & e^{\beta t_N}\end{bmatrix}\cdot\begin{bmatrix}a\\b\\c\end{bmatrix}=\begin{bmatrix}y_1\\y_2\\ \vdots\\y_N\end{bmatrix}$$

在上面的公式中，a、b、c和前一节的多项式一样，也是呈线性变化的。因此，我们在 Matlab 中就可以使用左除运算符求未知向量的最小二乘解。也就是说，如果我们用E表示最左边$N\times3$的矩阵，p表示向量$[a\ b\ c]'$，y表示右边的向量，则$p=E\backslash y$将给出p的最小二乘解。

当α和β为未知数时，上式不存在线性最小二乘解，这是因为在上述方程中，α和β是不能截然分开的。在这种情况下，我们有必要使用一个最小化算法，例如 Matlab 中的 fminsearch 函数。我们有两种方法建立通过 fminsearch 求解的模型。一种方法是将$f(t)$写为：

$$f(t)=x_3+x_4e^{x_1t}+x_5e^{x_2t}$$

上式中，我们将所有的未知数组合到了一个向量中，即$x=[x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5]'$。利用上面的公式，对于给定的一组数据$\{t_i, y_i\}, i=1,2,\cdots,N$，最小二乘解将使数据点$y_i$和函数在$t_i$处的估计值$f(t_i)$之间的范数最小。也就是说，如果$y$是一个包含$y_i$数据点的向量，$f$是一个包含$t_i$时刻的函数估计值的向量，则最小二乘解将由下式给出：

$$\min_x \|y-f\|$$

为了使用 fminsearch 函数，我们必须先求出上面的范数$\|y-f\|$，这一操作可以用下面的 M 函数文件完成：

```
function enorm=fitfun1(x,tdata,ydata)
%ENORM Norm of fit to example nonlinear function
% f(t) = x(3)+x(4)*exp(x(1)*t)+x(5)*exp(x(2)*t)
%
% ENORM(X,Tdata,Ydata) returns norm(Ydata-f(Tdata))

f=x(3)+x(4)*exp(x(1)*tdata)+x(5)*exp(x(2)*tdata);

enorm=norm(f-ydata);
```

下面给出的 M 脚本文件是一个完整的求解过程。该文件中创建了一些数据，作出了一个初始假设，并调用 fminsearch 函数来寻找一个解。

```
% testfitfun1
% script file to test nonlinear least squares problem
% create test data
x1 = -2; % alpha
x2 = -5; % beta
x3 = 10;
x4 = -4;
x5 = -6;
tdata = linspace(0,4,30)';
ydata = x3+x4*exp(x1*tdata)+x5*exp(x2*tdata);
% create an initial guess
x0 = zeros(5,1); % not a good one, but a common first guess
% call fminsearch
fitfun = @fitfun1; % create handle
options = []; % take default options
x = fminsearch(fitfun,x0,options,tdata,ydata)
% compute error norm at returned solution
enorm = fitfun(x,tdata,ydata)
```

运行上面的代码，结果如下：

```
>> testfitfun1
Exiting: Maximum number of function evaluation has been exceeded
         - increase MaxFunEvals option.
         Current function value: 3.234558
x =
    0.17368
    -1.5111
    13.972
    -2.2059
    -9.4466
enorm =
    3.2346
```

由结果可知，根据上面的初始假设，算法没有收敛。将函数的计算次数和算法迭代次数增加到2000，可以得到如下的结果：

```
>> options = optimset('MaxFunEvals',2e3,'MaxIter',2e3);
x = fminsearch(fitfun,x0,options,tdata,ydata)
x =
    -0.78361
    -3.7185
    10.061
    -0.80533
    -9.1907
enorm =
    0.23727
```

现在，算法是收敛的，但还没有收敛到x=[-2 -5 10 -4 -6]'处。下面的代码给出了曲线拟合更详细的信息，包括原始数据的曲线图，实际的函数，以及拟合的解：

```
% testfitfun2
% script file to test nonlinear least squares problem

% create test data
x1 = -2; % alpha
x2 = -5; % beta
x3 = 10;
x4 = -4;
x5 = -6;

tdata = linspace(0,4,30)';
ydata = x3+x4*exp(x1*tdata)+x5*exp(x2*tdata);

% create an initial guess

x0 = zeros(5,1); % not a good one, but a common first guess

% call fminsearch

fitfun = @fitfun1; % create handle

options = optimset('MaxFunEvals',2e3,'MaxIter',2e3);

x = fminsearch(fitfun,x0,options,tdata,ydata);

ti = linspace(0,4); % evaluation points
actual = x3+x4*exp(x1*ti)+x5*exp(x2*ti); % actual function

fitted = x(3)+x(4)*exp(x(1)*ti)+x(5)*exp(x(2)*ti); % fitted solution
subplot(2,1,1)
plot(tdata,ydata,'o',ti,actual,ti,fitted)
xlabel t
title 'Figure 38.5: Nonlinear Curve Fit'

subplot(2,1,2)
plot(ti,actual-fitted)
xlabel t
ylabel Error
```

运行上述代码，可得出下面的图形：

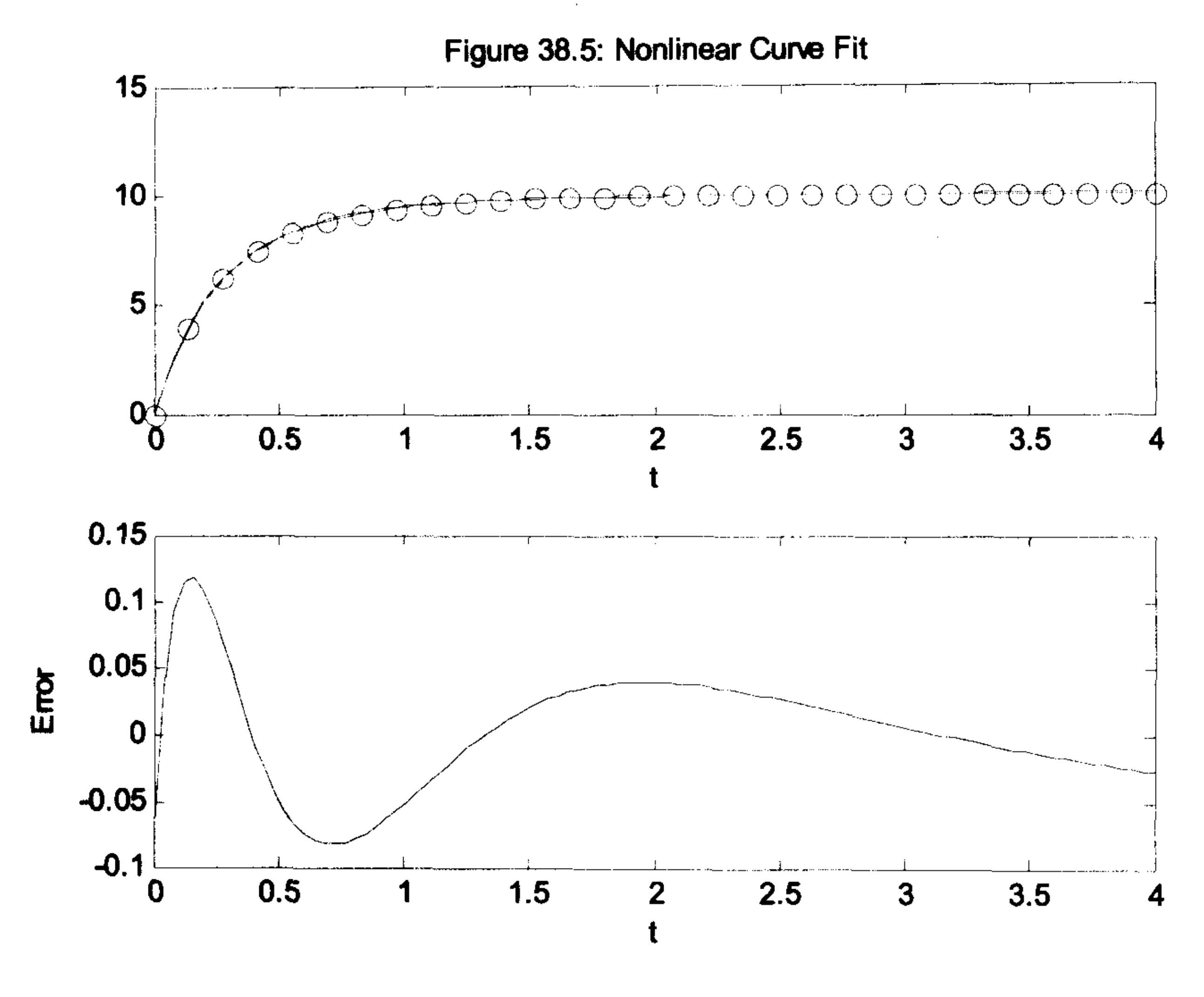


图 38.5 非线性曲线拟合

由上图可以看到，拟合解（以'o'标示的数据）的视觉效果还是相当不错的。不过，再看下面的误差图就会发现解的误差也是相当大的。如果上述问题的主要目标是使实际系数与 fminsearch 函数返回值之间的误差最小，那么上述算法显然是不符合要求的，尽管我们从主观角度看不出什么大的问题。

鉴于上述原因，我们就必须找出满足要求的解决方案。如果我们将收敛性判别标准（Convergence Criterion）进行更严格的设置，如下例所示：

```
>> options=optimset('TolX',1e-10,'TolFun',1e-10,'MaxFunEvals',
                        2e3,'MaxIter',2e3);
>> x = fminsearch(fitfun,x0,options,tdata,ydata)
x =
    -0.78359
    -3.7185
    10.061
    -0.80533
    -9.1907
    >> enorm = fitfun(x,tdata,ydata)
enorm =
    0.23727
```

我们发现，解并没有发生变化。

我们也可以使用更接近解的初始假设，如下面的代码所示：

```
>> x0 = [-1 -4 8 -3 -7]'; % much closer initial guess
>> options = []; % default options
>> x = fminsearch(fitfun,x0,options,tdata,ydata)
```

```
x =
    -2
    -5
    10
    -4
    -6
>> enorm = fitfun(x,tdata,ydata)
enorm =
    3.2652e-006
```

这时，fminsearch 函数返回的系数与用于创建数据的系数十分接近。该例充分说明了非线性优化算法固有的模糊性（Ambiguity）。也就是说，除了令误差范数为零外，再也没有获知解何时能够达到用户期望的确定性方法。

当一个非线性曲线拟合问题既包含线性部分，又包含非线性部分时（比如前面给出的例子），我们可以使用线性最小二乘法计算线性部分的解，使用诸如 fminsearch 这样的函数计算非线性部分的解。在大部分情况下，这一方法都能取得良好的效果。

为了更好地说明，下面我们来看一个例子。假设我们仍考虑前面给出的矩阵方程：

$$\begin{bmatrix} 1 & e^{\alpha t_1} & e^{\beta t_1} \\ 1 & e^{\alpha t_2} & e^{\beta t_2} \\ \vdots & \vdots & \vdots \\ 1 & e^{\alpha t_N} & e^{\beta t_N} \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

其中，α 和 β 都是已知常数。

现在，我们将 α 和 β 也作为 fminsearch 函数需要处理的变量。在函数体内部，当计算误差范数时，线性最小二乘问题将使用当前对 α 和 β 的估计值来求解。也就是说，如果 $x_1=\alpha$ ，$x_2=\beta$ ，p=[a b c]'，则我们可以将 fitfun1 函数重写如下：

```
function [enorm,p]=fitfun2(x,tdata,ydata)
%ENORM Norm of fit to example nonlinear function
% f(t) = p(1)+p(2)*exp(x(1)*t)+p(3)*exp(x(2)*t)
%
% ENORM(X,Tdata,Ydata) returns norm(Ydata-f(Tdata))
%
% [e,p]=ENORM(...) returns the linear least squares
%                  parameter vector p

% solve linear least squares problem given x input supplied by fminsearch

E = [ones(size(tdata)) exp(x(1)*tdata) exp(x(2)*tdata)];
p = E\ydata; % least squares solution for p=[a b c]'

% use p vector to compute error norm
f = p(1)+p(2)*exp(x(1)*tdata)+p(3)*exp(x(2)*tdata);

enorm = norm(f-ydata);
```

现在，fminsearch 函数只需要处理两个参数就可以了。下面的代码对上述函数进行了测试：

```
% testfitfun3
% script file to test nonlinear least squares problem

% create test data
x1 = -2; % alpha
x2 = -5; % beta
p1 = 10;
p2 = -4;
p3 = -6;

tdata = linspace(0,4,30)';
ydata = p1+p2*exp(x1*tdata)+p3*exp(x2*tdata);

% create an initial guess

x0 = zeros(2,1); % only two parameters to guess now!

% call fminsearch

fitfun = @fitfun2; % create handle to new function

options = []; % take default options

x = fminsearch(fitfun,x0,options,tdata,ydata)

% find p and compute error norm at returned solution

[enorm,p] = fitfun(x,tdata,ydata)
```

运行上面的程序，结果如下：

```
>> testfitfun3
x =
    -2
    -5
enorm =
    7.1427e-006
p =
    10
    -4
    -6
```

从上述结果可以看出，fminsearch 基本上不费什么周折就得到了与实际值非常接近的解。将线性求解和非线性求解结合在一起，可以大大降低搜索空间的维数，同时也会大大改进非线性算法的速度和收敛性。例如，在上面的例子中，搜索空间的维数就由五维降到了二维。

最后，用户需要注意 p 是如何作为 fitfun2 函数的第二个输出参数使用的。在计算过程中，最小化算法函数 fminsearch 将忽略此参数，因此，该参数对搜索过程不产生任何影响。不过，当算法结束时，fitfun2 函数将被再次调用，此时使用了两个输出参数用于返回线性系数的解。注意，全局变量最好不要用于保存 p 变量，因为我们无法知道 fminsearch 函数是否在最后一次迭代时，在求出的解处执行了 fitfun2。在 fminsearch 结束后调用 fitfun2 可以保证系数向量 p 是在 fminsearch 函数求出的解处被执行的。

38.11 画中画缩放

最后一个例子向大家阐述 Matlab 中句柄图形特性的用法。从示意图 38.6 可以看出，该例实现了画中画缩放的功能。也就是说，如果用户调用了本节所提供的缩放函数，然后在当前的坐标轴中用鼠标选择一个矩形区域，该区域将被保持为一个由点线框起来的矩形，并且会生成一个新的稍小一些的坐标轴用于显示选择的图形部分。可见，该函数使用户可以在不隐藏原来图形的情况下，对图形的局部进行放大。另外，小坐标轴还可以被选择、拖动和改变大小。

要实现上述功能需要一些步骤，我们无法也没必要将这些步骤写在一个巨大的函数中。相反，将这些步骤进行分割，用一些小型的支持函数表示出来，并作为主函数的子函数来调用，或者将它们放在 Matlab 的搜索路径中直接进行调用，是一个简单易行且思路清晰的方法。采用这种方法，我们需要创建的第一个函数是 getn，该函数仅仅将 get 函数的输出参数简化为单个变量，其代码如下所示：

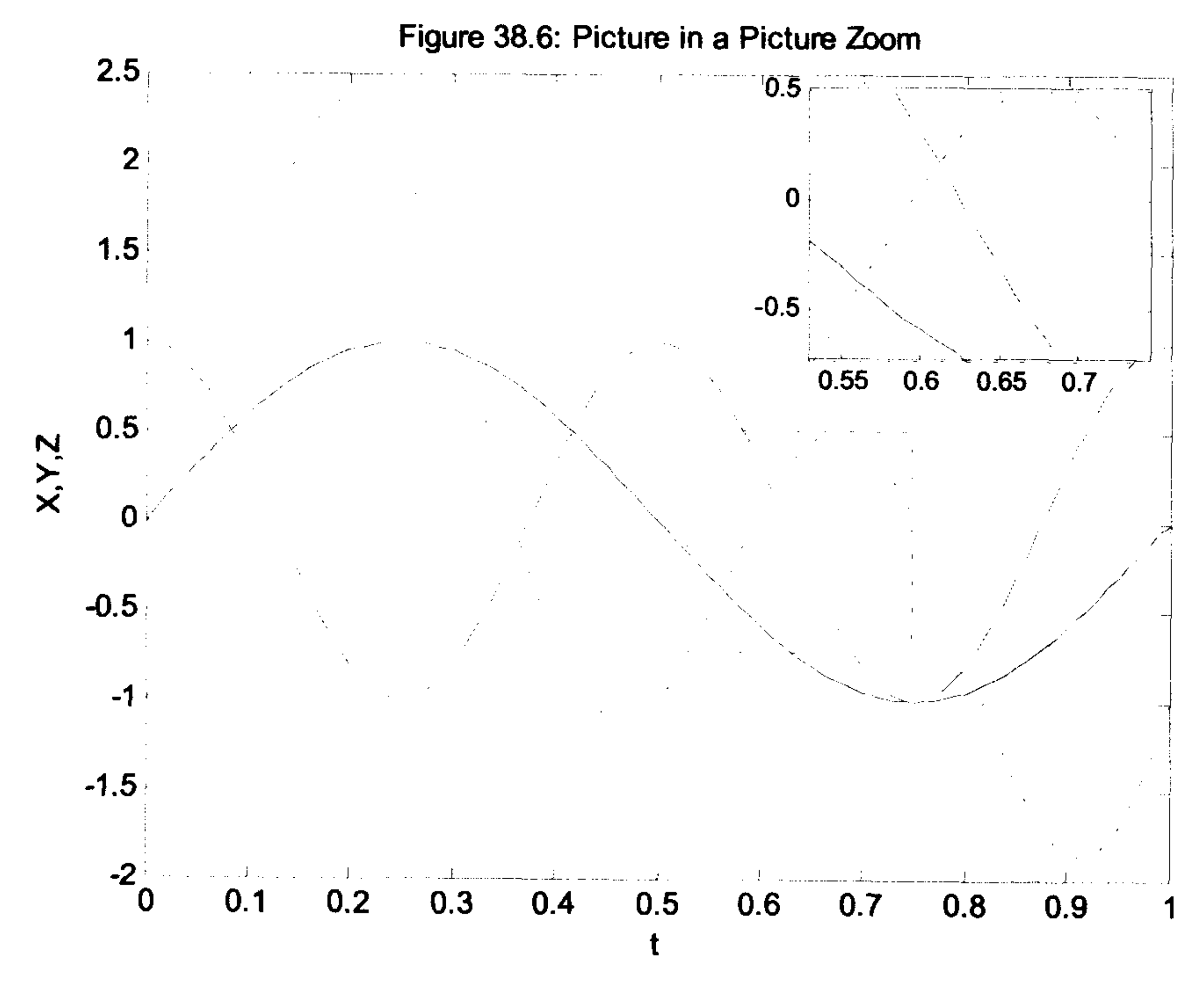


图 38.6　画中画缩放

```
function varargout=getn(H,varargin)
%GETN Get Multiple Object Properties.
% [Prop1,Prop2,...] = GETN(H,PName1,PName2,...) returns the requested
% properties of the scalar handle H in the corresponding output arguments.
%
% For example, [Xlim,Ylim,Xlabel] = GETN(gca,'Xlim','Ylim','Xlabel')
```

```
% returns the requested axes properties in like-named output variables.
%
% This simplifies the construct
% [Xlim,Ylim,Xlabel] = deal(get(gca,{'Xlim','Ylim','Xlabel'}))

if max(size(H))~=1 || ~ishandle(H)
  error('Scalar Object Handle Required.')
end
varargout=get(H,varargin);
```

getn 表面上看起来用处不大，但它会给我们后面的处理带来方便。同时，它向读者演示了 varargin 和 varargout 参数的使用方法。

下面要考虑的一个函数是如何实现矩形区域的选择，并捕捉该区域的纵横坐标（即 x 轴和 y 轴）的范围。下面给出了这一步的实现函数，其中用到了 Matlab 函数 rbbox 和 waitforbuttonpress，同时还用到了坐标轴对象和图形对象的 CurrentPoint 属性：

```
function [xbox,ybox,prect]=getbox
%GETBOX Get axes information from user-drawn selection rectangle.
% [Xbox,Ybox,Prect]=GETBOX waits for the user to drag a selection box The
% x and y axis data limits of the selection box are returned in Xbox, Ybox,
% and Prect.
% Xbox is a two element vector containing the minimum and maximum limits
% along the x axis, i.e., Xbox = [min(x) max(x)]
% Ybox is a two element vector containing the minimum and maximum limits
% along the y axis, i.e., Ybox = [min(y) max(y)]
% Prect is a four element vector containing the selection box position
% in standard position vector format, Prect = [left bottom width height]
% Data returned is in the axis data units.
%
% The selection box is limited to the x and y axis limits of the axes where
% the selection rectangle was drawn.

% waitforbuttonpress waits until user presses a mouse button over a figure
% waitforbuttonpress return False if that happens. Alternatively, it
% returns True if the user presses a key on the keyboard.

if waitforbuttonpress % Returns True if a key is pressed, abort
  xlim=[];
  ylim=[];
  prect=[];
  return
end
% Function only gets here if user presses a mouse button in a figure

Hf = gcf;      % get current figure where button was pressed
Ha = gca(Hf);  % get current axes where button was pressed

AxesPt = get(Ha,'CurrentPoint'); % get first axes data point clicked
FigPt = get(Hf,'CurrentPoint');  % get first figure point clicked
```

```
% call the function rbbox, i.e., rubberband box, to create the selection
% rectangle. This function needs to know where to start from. It does not
% automatically start at the mouse click unless told to do so.

% drag selection rectangle starting at first figure point
rbbox([FigPt 0 0],FigPt) % function returns as soon as mouse button is up

% get point on opposite corner of selection rectangle; add to first point
AxesPt = [AxesPt;get(Ha,'CurrentPoint')];

% get axis limits of axes where selection rectangle was drawn
[Xlim,Ylim] = getn(Ha,'Xlim','Ylim');

% convert AxesPt data into usable output vectors.
xbox = [min(AxesPt(:,1)) max(AxesPt(:,1))]; % x axis limits of selection
xbox = [max(xbox(1),Xlim(1)) min(xbox(2),Xlim(2))]; % limit to axes size

ybox = [min(AxesPt(:,2)) max(AxesPt(:,2))];
ybox = [max(ybox(1),Ylim(1)) min(ybox(2),Ylim(2))]; % limit to axes size

prect = [xbox(1) ybox(1) diff(xbox) diff(ybox)]; % position rectangle
```

getbox 函数对每一步操作都进行了详细注释。有一点需要注意，那就是坐标轴对象的 CurrentPoint 属性将返回一个 2×3 的矩阵，用以表示三维空间中当前点前后两点的(x,y,z)坐标值。但由于我们考虑的是一个二维坐标，因此，我们只关心矩阵的前两列。对这两列数据进行处理，就得到了我们需要的输出变量。

使用 getbox 和 getn 函数，实现画中画缩放的主函数 mmzoom 如下所示：

```
function [Hza,Hzr]=mmzoom(arg)
%MMZOOM Picture in a Picture Zoom.
% MMZOOM creates a new axes containing the data inside a box formed by a
% click and drag with the mouse in the current axes. The new zoomed axes
% is placed in the upper right of the current axes, but can be moved with
% the mouse. Clicking in the figure border disables dragging.
%
% Previous axes created by MMZOOM are deleted if MMZOOM is called again.
%
% [Hza,Hzr] = MMZOOM returns handles to the created axes and rectangle
% marking the zoomed area respectively.
%
% MMZOOM DRAG enables dragging of a zoomed axes.
% MMZOOM RESET disables dragging of a zoomed axes.
% MMZOOM OFF removes the zoomed axes and rectangle marking the zoomed area.

if nargin==0
  arg = [];
end
if isempty(arg) % zoom zoom zoom zoom zoom zoom zoom zoom zoom zoom zoom

  Hzoom = findobj(0,'Tag','MMZOOM'); % find previous zoomed axes
  if ~isempty(Hzoom)  % delete prior zoomed axes if it exists
```

```
    delete(Hzoom)
end

[xlim,ylim,prect] = getbox;  % get selection box for zoom

if ~isempty(prect) % act only if rectangle exists
    Haxes = gca; % handle of axes where selection box was drawn
    Hzr = rectangle('Position',prect,... % place rectangle object
        'Linestyle',':',...                % to mark selection box
        'Tag','MMZOOM');

    Hfig = gcf; % handle of Figure where selection box was drawn
    Hzoom = copyobj(Haxes,Hfig); % copy original axes and its children

    OldUnits = get(Haxes,'Units');  % get position vector of original
    set(Haxes,'Units','normalized') % axes in normalized units
    Pvect = get(Haxes,'Position');
    set(Haxes,'Units',OldUnits)

    % scale and shift zoomed axes relative to original axes
    alpha = 1/3;   % position scaling for zoomed axes
    beta = 98/100; % position shift for zoomed axes

    % compute position vector for zoomed axes
    Zwidth = alpha*Pvect(3);                      % zoomed axes width
    Zheight = alpha*Pvect(4);                     % zoomed axes height
    Zleft = Pvect(1)+beta*Pvect(3)-Zwidth;        % zoomed axes left
    Zbottom = Pvect(2)+beta*Pvect(4)-Zheight;     % zoomed axes bottom

    % modify zoomed axes as required
    set(Hzoom,'units','Normalized',...           % make units normalized
        'Position',[Zleft Zbottom Zwidth Zheight],...% axes position
        'Xlim',xlim,'Ylim',ylim,...              % axis data limits
        'Box','on',...                            % axis box on
        'Xgrid','off','Ygrid','off',...           % grid lines off
        'FontUnits','points',...
        'FontSize',8,...                          % shrink font size
        'ButtonDownFcn',@selectmoveresize,...     % enable drag
        'Tag','MMZOOM',...                        % tag zoomed axes
        'UserData',Haxes)                         % store original axes

    [Htx,Hty,Htt] = getn(Hzoom,'Xlabel','Ylabel','Title');
    set([Htx,Hty,Htt],'String','')      % delete labels on zoomed axes

    set(Haxes,'DeleteFcn',...           % delete both axes together
        'delete(findobj(0,''Type'',''axes'',''Tag'',''MMZOOM''))')

    % place zoomed axes at top of object stack
    Hchild = findobj(Hfig,'type','axes'); % get all axes in figure
    Hchild(Hchild==Hzoom) = [];           % remove zoomed axes from list

    set(Hfig,'Children',[Hzoom;Hchild],...% put zoom axes at top of stack
        'CurrentAxes',Haxes,...             % make original axes current
        'ButtonDownFcn','mmzoom reset')    % enable reset
```

```
      if nargout>=1  % provide output only if requested
         Hza = Hzoom;
      end
   end

elseif strncmpi(arg,'d',1) % drag zoom axes drag zoom axes drag zoom axes

   Hzoom = findobj(0,'Type','axes','Tag','MMZOOM');
   if ~isempty(Hzoom)
      set(Hzoom,'ButtonDownFcn',@selectmoveresize)
   end

elseif strncmpi(arg,'r',1) % reset reset reset reset reset reset reset

   Hzoom = findobj(0,'Type','axes','Tag','MMZOOM');
   if ~isempty(Hzoom)
      [Hfig,Haxes] = getn(Hzoom,'Parent','UserData');
      set(Hzoom,'ButtonDownFcn','','Selected','off')% turn off selection
      set(Hfig,'CurrentAxes',Haxes)                  % make Haxes current
   end

elseif strncmpi(arg,'o',1)  % off off off off off off off off off off off

   Hzoom = findobj(0,'Tag','MMZOOM');
   if ~isempty(Hzoom)
      delete(Hzoom)
   end
else
   error('Unknown Input Argument.')
end
```

mmzoom 函数使用了切换结构来根据输入参数执行代码。如果输入参数没有使用，则 mmzoom 将创建一个缩放坐标轴。其中，函数 copyobj 用于制作一个原始坐标轴对象及其子对象的副本。然后对这个副本进行修改来创建缩放坐标轴。最后将缩放坐标轴放置在原坐标轴的顶部，并设置为当前坐标轴。缩放坐标轴的 ButtonDownFcn 回调将执行 selectmoveresize 函数，执行对该坐标轴的选择、移动和改变大小操作。图形对象的 ButtonDownFcn 回调则调用 mmzoom('reset')，用于删除缩放坐标轴对象及其回调。mmzoom 函数中将使用 findobj 函数寻找缩放坐标轴和选择的矩形区域。不过，使用 findobj 函数就无法同时创建多个 mmzoom 的实例，即无法同时创建多个缩放坐标轴。如果用户需要创建多个缩放坐标轴，mmzoom 的函数句柄回调中就需要添加另外的参数，具体实现方法请参考本书第 32 章，这里不做详述。

附录

Matlab 版本信息

本附录列举了自 Matlab 5 以来，Maltab 各发行版本增加和修改的功能和函数，有助于用户区别 Matlab 各个版本都有哪些特有的功能和函数。目前，Matlab 已成为全世界广泛使用的大众软件，并且各个版本都拥有众多的用户，加之当今文件共享越来越广泛和容易，因此，我们有必要帮助读者正确理解 Matlab 的 M 文件在各个版本之间的上下兼容性。本附录将 Matlab 各版本的信息集中在一起，能够帮助读者正确使用 Matlab 的功能和函数，这在其他文献中是找不到的。

我们给出的每个版本的信息，主要包括公共发布的日期、新增的功能和函数，以及修改的功能和函数。另外，一些不再使用的旧体函数也被搜集了进来。对于修改过的功能和函数，我们将在这些功能和函数的描述的最前面用“（修改）”这两个字标示。如果所列举的功能和函数在本书出现过，我们还给出了它们出现在本书的章节。

本附录所列举的材料都来源于基本的 Matlab 产品和 Matlab 各版本的版本注释中的信息，不包含各种工具箱的内容或第三方提供的附加功能和函数。不过，由于版本注释并不很容易理解，一些信息可能会被忽略。还有，由于句柄图形对象及其属性所包含的信息太多，因此，本附录中也没有列举这些内容。

Matlab 5.0（发布日期：1996 年 12 月）

功能	描述	出现章节
多维数组	具有任意维数的数组（例如，A(i,j,k,...)）	6
结构体	一种可以装载变量的类，由一个变量名和若干字段名表示，其中字段可以通过句点符访问，如，varname.field1	8
单元数组	一种可以装载变量的类，由一个变量名表示，其内容通常被包含在一对花括号（{}）中	8
面向对象编程	一种用户定义的变量类，通常包含函数和操作符重载、数据封装、方法、继承等等	33
子函数	一种包含多个函数的 M 文件函数，其中出现在第一个函数（又称为主函数）后面的函数都称为子函数	12
私有函数	保存在 Matlab 路径下的 private 子目录中的 M 函数文件	12

（续表）

功能	描述	出现章节
Switch/case 语句	选择控制流程结构	11
变量长度函数的输入和输出列表	函数输入和输出参数列表支持的单元数组 varargin 和 varargout	12
伪码创建	将 M 文件预编译成一种加密格式的能力	12
图像的单字节数据类型	支持 8 比特整数数据的图像对象，即使用 uint8 数据类型	7，29
真彩色	支持用真彩色表示的图像	28
用 end 表示数组的最后一个元素	某一维的最后一个元素可以使用关键字 end 来寻址，（例如，A(1:2:end)）	5
标量扩展赋值	（修改）对于语句 A(…)=x，将使用标量扩展用标量 x 填充所有指定的 A 中的元素	5
字符串存储	（修改）每一个字符在存储时占用 2 个字节，而不是 8 个	9
在数据分析函数中指定要分析的维	（修改）诸如 sum、prod、cumprod 和 cumsum 都可以接受一个用于指定所处理的维的输入参数（通常为最后一个参数）	18
空数组	（修改）空数组也可以具有非零长度的维	5
标记类型的强化	（修改）支持新的直线标记，并且标记可以被单独指定	26
摄像机视角模型	坐标轴新增了摄像机属性，用于设置视角	27
TeX 支持	（修改）文本对象可以包含 TeX 命令	31
Z 缓冲	支持 Z 缓冲图形渲染	28
光照对象	一种新增的图形对象，用于设置灯光效果	31

数学函数	描述
airy	Airy 函数
besselh	第三类贝塞尔函数（即 Hankel 函数）
condeig	特征值的条件数
condest	估计方阵的 1 范数矩阵条件数
dblquad	数值二维积分
mod	求一个复数的模
normest	2 范数估计

n 维函数	描述
cat	数组串联
flipdim	沿着指定的维翻转数组
ndgrid	产生 n 维函数所需的数组
ndims	数组的维的个数
permute、ipermute	对 n 维数组的各维进行重排，以及对重排的维进行复原

（续表）

n 维函数	描述
reshape	改变数组的形状
shiftdim	对数组的维进行循环平移
squeeze	删除单一维
sub2ind、ind2sub	在数组的单值索引和下标之间进行转化

单元数组和结构体函数	描述
cell	创建单元数组
cell2struct	将单元数组转换为结构体
celldisp	显示单元结构
cellplot	利用图形显示单元结构
num2cell	将矩阵转换为单元数组
fieldnames	结构体的字段名
getfield	从结构体中获取字段
setfield	将字段从结构体删除
struct	创建结构体数组
struct2cell	将结构体转换为单元数组

字符函数	描述
char	转化为字符串数组
mat2str	将矩阵转化为字符串数组
strcat	字符串串联
strmatch	选择与指定字符匹配的字符串
strncmp	比较两个字符串的前 n 个字符
strvcat	纵向字符串串联

逻辑函数	描述
iscell	判断是否为单元数组
isequal	判断数组是否相等
isfinite	判断数组是否具有无限多个元素
islogical	判断是否为逻辑数组
isnumeric	判断是否为数值数组
isstruct	判断是否为结构体
logical	将数组转换为逻辑数组

M 文件处理函数	描述
assignin	为工作区中的变量赋值
evalin	在工作区中执行一条表达式
inmem	内存中的函数
inputname	输入参数的名称
mfilename	当前正在运行的 M 文件名
mexext	MEX 文件扩展名
pcode	创建伪码
profile	剖析 M 文件的执行
varargin、varargout	传递或返回函数参数的可变个数
warning	显示告警消息

文件和路径函数	描述
addpath	在 Matlab 搜索路径列表中添加目录
edit	编辑 M 文件
editpath	修改 Matlab 搜索路径列表
fullfile	从部分名称中创建文件的全名

集合、位和基底函数	描述
intersect	设置交集
ismember	检查是否是一个集合的成员
setdiff	设置差集
setxor	设置异或集
union	合并两个集合
unique	去除一个向量中的重复元素
bitand	基于比特的与操作
bitcmp	逐比特比较
bitget	获取一个比特
bitmax	浮点整数的最大值
bitor	基于比特的或操作
bitset	设置比特
bitshift	基于比特循环移位
bitxor	基于比特的异或操作
base2dec	将基底转换为十进制
bin2dec	将二进制转换为十进制
dec2base	将十进制转换为基底
dec2bin	将十进制转换为二进制

时间和日期函数	描述
calendar	创建日历
datenum	计算日期数
datestr	创建日期字符串
datetick	创建日期格式的坐标轴标注
datevec	将日期分解成各个成分
eomday	返回一个月的最后一天
now	显示当前时间和日期
weekday	显示星期

矩阵函数	描述
cholinc	非完备 Cholesky 因式分解
gallery	一个超过 50 个测试矩阵的矩阵集合
luinc	非完备 LU 因式分解
repmat	复制和排列数组
sprand	创建均匀分布的随机稀疏数组

稀疏矩阵函数	描述
bicg	双共轭梯度方法
bicgstab	双共轭梯度稳定方法
cgs	共轭梯度乘方方法
eigs	寻找几个特征值和特征向量
gmres	广义最小参差方法
pcg	预处理共轭梯度方法
qmr	准最小参差方法
svds	创建奇异值

数据分析函数	描述
convhull	创建凸表面
cumtrapz	累积梯形数值积分
delaunay	Delaunay 三角化
dsearch	寻找最近的点
factor	生成素因子
inpolygon	判断一个点是否在某一多边形区域中
isprime	判断一个数是否为素数
nchoosek	返回从 n 个元素提取 k 个元素的所有可能的组合

（续表）

数据分析函数	描述
perms	返回所有的排列组合
polyarea	求多边形的面积
primes	创建一个素数列表
sortrows	将行按升序排列
tsearch	寻找封闭的 Delaunay 三角形
voronoi	创建 Voronoi 图表
sum([ ])=0、prod([ ])=1、max([ ])=[ ]、min([ ])=[ ]	（修改）当输入为空数组时，定义函数的输出

插值函数	描述
interp3	三维数据插值
interpn	N 维数据插值
ndgrid	创建可供 N 维函数和 N 维插值使用的数组
griddata	（修改）使用基于三角形的插值

ODE 函数	描述
ode45、ode23、ode113、ode23s、ode15s	使用不同的方法求解差分方程
odefile	用于 ODE 求解算子的问题定义文件
odeget	从 ODE 选项结构中提取选项
odeset	创建或修改 ODE 求解算子的选项结构

绘图函数	描述
area	绘制填充的区域
bar3	绘制三维条状图
bar3h	绘制三维横向条状图
barh	绘制二维横向条状图
box	显示或隐藏坐标轴框
countourf	绘制填充的等高线
pie	绘制饼图
pie3	绘制三维饼图
plotyy	绘制具有左右两个纵轴的图形
quiver3	绘制三维箭头图
ribbon	将线绘制成三维条形
stem	（修改）棒状图的顶端既可以填充，也可以不填充
stem3	绘制三维棒状图
trimesh	绘制三角网状图
trisurf	绘制三角表面图

颜色和光照函数	描述
autumn	创建具有红色和黄色的颜色映射表
colorcube	创建均匀分布的颜色映射表
colordef	选择图形颜色方式
lines	创建遵循坐标轴 colororder 属性的颜色表
spring	创建具有洋红色和黄色的颜色映射表
summer	创建具有绿色和黄色的颜色映射表
winter	创建具有蓝色和绿色的颜色映射表
imread	读取图像数据
imwrite	将图像数据保存到文件

句柄图形和 GUI 函数	描述
dragrect	创建用户用鼠标拖动选择的矩形区域
inputdlg	显示输入对话框
msgbox	显示消息对话框
questdlg	显示询问对话框
rbbox	创建“橡皮圈”图框
selectmoveresize	交互式选择、拖动和改变对象的大小
uiresume	恢复被挂起的 M 文件执行
uiwait	停止一个 M 文件的执行
waitfor	停止 M 文件执行，直到满足一定的条件

Matlab 5.1（发布日期：1997 年 6 月）

功能	描述	出现章节
find 函数可以返回空数组	（修改）如果没有找到使参数为真的索引值，find(…)就返回[ ]	5
多字节字符	（修改）支持两字节字符	9

函数	描述
ismember	在本版本中，此函数内部将调用 MEX 函数 ismemc 来使执行速度最大化
pagedlg	打开一个页面设置对话框
printdlg	打开一个打印对话框
scatter	二维分散点绘图
scatter3	三维分散点绘图

Matlab 5.2 R10（发布日期：1998 年 3 月）

功能	描述	出现章节
try/catch 模块	使用 try/catch 模块进行错误控制	11
M 文件锁定	当 M 文件被锁定后，clear 函数就无法将该 M 文件从内存清除	12
永久（Persistent）变量	将变量声明为永久变量，这样，该变量在被函数调用期间会一直存在	12
S=load(…)	（修改）将文件中的内容载入到结构体中	14
HDF 文件支持	可以读写 HDF 格式的文件	14
OpenGL 支持	支持 OpenGL 图形渲染	28
工具提示	UI 控件将具有一个 tooltip 属性	32
开关按钮	一个新的 Uicontrol 对象	32
Uicontextmenu 对象	一个新的 Uimenu 对象	32
支持字符串单元数组	在此版本中，函数 intersect、ismember、lower、setdiff、setxor、sort、union、unique 和 upper 都支持字符串单元数组输入参数	9

函数	描述
camdolly	变换摄像机的位置和目标
camlight	在摄像机坐标系统中创建或移动光照对象
camorbit	以摄像机目标为中心旋转摄像机位置
campan	以摄像机位置为中心旋转摄像机目标
camroll	以摄像机视线为中心旋转摄像机
camzoom	调整摄像机焦距
campos	设置或获取摄像机位置和位置模式
camproj	设置或获取摄像机投影类型
camtarget	设置或获取摄像机目标和目标模式
camup	设置或获取摄像机向上矢量（up vector）和向上矢量模式
camva	设置或获取摄像机视角和视角模式
cholinc	稀疏非完备 Cholesky 和无限 Cholesky 因式分解
cholupdate	将秩为 1 的矩阵升级为 Cholesky 因式分解
daspect	设置或获取数据的纵横比或纵横比模式
ifftshift	逆 FFT 的循环移位
lastwarn	返回最后一个告警字符串
lightangle	在球坐标系统中创建或移动光照对象
mislocked	如果 M 文件不能被清除，则为 True
mlock	保护一个 M 文件不被清除
munlock	允许一个 M 文件被清除

（续表）

函数	描述
ode23t	用于求解中等难度的差分方程
ode23tb	使用较宽松的误差容限求解有难度的差分方程
pbaspect	设置或获取绘图框的纵横比或纵横比模式
persistent	将一个变量声明为永久变量
strcmpi	用于字符串比较，忽略大小写
strncmpi	比较两个字符串中的前 n 个字符，忽略大小写
strjust	（修改）现在支持左调整、右调整和中心调整
xlim	设置或获取 x 轴的坐标范围和范围模式
ylim	设置或获取 y 轴的坐标范围和范围模式
zlim	设置或获取 z 轴的坐标范围和范围模式

Matlab 5.3 R11（发布日期：1999 年 3 月）

功能	描述	出现章节
整数数据类型	增加了对 int8、int16、uint16、int32 和 uint32 等数据类型的支持	7
单精度数据	增加了用单精度（single）存储数据的功能	7
退出时执行某个函数	当 Matlab 退出时，Matlab 会执行函数 finish	4
支持大型矩阵	ODE 组件中的所有函数都支持使用大型矩阵（mass matrix）	25
优化参数	用户可以使用一个选项结构体变量，而非一个选项向量来设置一些优化函数（如 fminbnd、fminsearch 和 lsqnonneg）的选项	23
矩形对象	用户标绘矩形、椭圆和圆的一类新的句柄图形对象	31
双缓冲	图形窗口现在支持双缓冲数据存储计数，以降低数据闪烁现象	31
可移植网络图形	支持可移植网络图形（PNG）格式的图像	29

函数	描述
blkdiag	创建块对角矩阵
cellfun	对单元数组进行普通操作
complex	由实部和虚部创建复数数组
coneplot	在一个三维向量域中绘制锥形的速度向量
contourslice	在一个体积截面上绘制等高线
datenum、datestr、datevec	（修改）现在这些函数接受 pivotyear 参数
evalc	字符串求值，其结果也被转化为字符串
ezcontour	简单的等高线绘制器
ezcontourf	简单的填充等高线绘制器
ezmesh	简单的网格图绘制器
ezmeshc	简单的带有等高线的网格图绘制器

（续表）

函数	描述
ezplot	简单的二维曲线绘制器
ezplot3	简单的三维曲线绘制器
ezpolar	简单的极坐标绘制器
ezsurf	简单的表面图绘制器
ezsurfc	简单的带有等高线的表面图绘制器
findfigs	寻找所有可见的图形窗口
fminbnd	（修改）fmin 函数的一个新名称
fminsearch	（修改）fmins 函数的一个新名称
hist	（修改）该函数内部调用了 MEX 函数 histc
histc	给定边界条件下的统计值计算，即计算一个数组中落入给定边界中的元素个数
int8、int16、uint16、int32、uint32	转换为整数数据类型
isocaps	计算等值面的顶面几何表面
isonormals	计算等值面顶点的法线
isosurface	从体积数据中提取等值面数据
lsqnonneg	（修改）nnls 函数的一个新名称
optimget、optimset	获取、设置和修改优化选项
pause	（修改）支持分数作为第二个参数
reducepatch	减少碎块表面的个数
reducevolume	减少体积数据元素的个数
shrinkfaces	减小碎片表面的大小
single	转换为单精度数据类型
smooth3	三维数据平滑
str2double	将字符串转换为双精度值
stream2	计算二维流水线
stream3	计算三维流水线
streamline	绘制流水线
subvolume	提取体积数据集的子集
sum	（修改）现在该函数可用于所有的整数数据类型
surf2patch	将表面数据转换为碎片数据
texlabel	由字符串创建 TeX 格式的数据

Matlab 6.0 R12（发布日期：2000 年 11 月）

功能	描述	出现章节
Matlab 桌面	一种用户接口，由 Java 语言实现	3
Java	Matlab 现在支持与 Java 语言的接口	
矩阵数学运算	（修改）在 BLAS 中，Matlab 将使用 LAPACK 进行矩阵运算	17

（续表）

功能	描述	出现章节
快速傅里叶变换	（修改）现在，Matlab 可以使用 MIT FFTW 库计算快速傅里叶变换	22
函数句柄	一种数据类型，用于捕捉被执行的函数信息	12
透明	（修改）现在，表面、碎片和图像都支持透明属性	29
操作符优先级	（修改）现在，逻辑与比逻辑或的优先级高	10

函数	描述
beep	使计算机发出“哔哔”声
numel	计算一个数组中的元素个数
continue	跳出剩余的 For 循环或 While 循环
genpath	产生一个包含一个指定路径中所有目录的字符串
sort	（修改）现在，该函数可以支持多种数据类型，而不仅仅是双精度值
std	（修改）现在，std([ ])能够返回 NaN，而不是空数组
polyfit、polyval	（修改）现在这两个函数支持数据的中心定位和按比例缩放

逻辑函数	描述
iskeyword	创建一个关键字，或检查一个参数是否为 Matlab 的关键字
isvarname	判断一个输入字符串是否是有效的变量名

M 文件处理函数	描述
check_syntactic_warnings	对 M 文件进行语法检查
func2str	根据函数句柄创建函数名字符串
functions	显示一个函数句柄的信息
nargoutchk	使输出参数的个数有效
rehash	更新函数和文件系统的缓存
str2func	根据函数名字符串创建函数句柄

稀疏矩阵函数	描述
colamd	计算一个稀疏矩阵基于列的近似最小级别的排列
lsqr	标准方程中共轭梯度的 LSQR 实现
minres	使用最小参差方法解方程组
symamd	系统近似最小级别排列
symmlq	使用系统 LQ 方法解方程组

插值函数	描述
convhulln	n 维凸包
delaunay3	三维 Delaunay 棋盘图
delaunayn	n 维 Delaunay 棋盘图

（续表）

插值函数	描述
desearchn	n 维最近点搜索
griddata3	三维数据栅格
griddatan	n 维数据栅格
interp1	（修改）现在，该函数可能使用'cubic'选项调用 pchip 函数
pchip	基于分段立方 Hermite 内插的多项式插值
voronoin	n 维 Voronoi 图表

优化或积分函数	描述
quadl	使用 Lobatto 积分法计算数值积分，该函数用于代替 quad8 函数
dblquad	（修改）现在可以通过额外的参数指定用于积分的函数
fzero	（修改）调用被改变的序列

差分方程函数	描述
bvp4c	使用排列求解两点边界值问题
bvpget	从选项结构体中获取 BVP 选项
bvpinit	创建 bvp4c 函数所需要的初始假设
bvpset	创建或改变 BVP 选项结构体
bvpval	评价 bvp4c 函数求出的解
ode	（修改）现在，即使不使用 ODE 文件，ODE 求解器也能求解问题
pdepe	在某一维上求解部分差分方程
pdeval	评价 pdepe 函数求出的解

三维可视化函数	描述
coneplot	绘制三维锥形
curl	在一个矢量域计算旋度和角速度垂线
divergence	计算矢量域的散度
interpstreamspeed	根据速度内插流线顶点
isocolors	计算等值面顶点的颜色
isosurface	提取等值面
streamparticles	绘制流粒子
streamribbon	绘制流带
streamslice	绘制流线
streamtube	绘制流体
volumebounds	获取体积数据的坐标范围和颜色范围

句柄图形和 GUI 函数	描述
alpha	设置或获取透明属性
alphamap	指定图形的透明度
alim	设置或获取坐标轴的α 范围

Java 函数	描述
import	向当前 Java 软件包导入列表中添加内容
isjava	判断是否为 Java 对象
javaArray	创建 Java 数组
javaMethod	激活 Java 方法
javaObject	创建 Java 对象
methodsview	显示 Java 或 Matlab 类执行的所有方法的信息

废弃不用的函数	描述
errortrap	已经被 try-catch 模块代替
flops	执行浮点操作计数
fmin	已经被 fminbnd 函数代替
fmins	已经被 fminsearch 函数代替
foptions	已经被 optimget 和 optimset 函数代替
interp4、interp5、interp6	已经被 interp2 函数代替
isdir	已经被 exist 函数代替
isiee	现在，所有的 Matlab 所在的操作系统都使用 IEEE 运算标准
isstr	已经被 ischar 函数代替
meshdom	已经被 meshgrid 函数代替
nnls	已经被 lsnonneg 函数代替
quad8	已经被 quad1 函数代替
saxis	已经不再使用此函数
setstr	已经被 char 函数代替
str2mat	已经被 char 函数代替
table1	已经被 interp1 函数代替
table2	已经被 interp2 函数代替

Matlab 6.1 R12.1（发布日期：2001 年 6 月）

功能	描述	出现章节
透明图例	现在，坐标轴的图例框可以被设置为透明状态	26

函数	描述
audioplayer	创建音频对象来播放 Windows 操作系统中的声音数据
audiorecorder	创建音频对象将声音数据录制在 Windows 操作系统中
bvpval	（废弃）现在已经被 deval 函数代替
convhull、delaunay、griddata、voronoi	（修改）现在，这些函数内部可以使用 Qhull 算法
cdfinfo	获取 CDF 文件信息
cdfread	读取 CDF 文件
convhull	（修改）现在，语句[K,a]=conhull(x,y)将返回凸曲面的面积，并且，该函数还忽略其第三个输入参数
convhulln	（修改）现在，语句[K,v]=conhulln(x,y)将返回凸曲面的体积
datenum、datestr	（修改）现在，这两个函数可以接受时间向量作为输入参数
delaunay	（修改）现在，该函数将忽略其第三个输入参数
deval	该函数用于执行 ODE 求解，另外，还用来代替已经过时的函数 bvpval
erfcinv	反向互补误差函数
fitsinfo	获取 FITS 文件的信息
fitsread	读取 FITS 文件
hdfinfo	获取 HDF 文件的信息
hdfread	读取 HDF 文件
histc	（修改）现在，该函数能够锁定执行时出错的位置
interp1	（修改）现在，该函数允许数据外插
numel	（修改）现在，语句 numel(A,varargin)将返回 A(varargin{:})中元素的个数
ode	（修改）现在，可以通过选项设置，返回 deval 需要使用的解结构
polyeig	（修改）现在，该函数只能返回特征值
ppval	（修改）现在，用户可以通过 ppval(xx,pp)的调用格式，实现 ppval 函数的自身嵌套
quad	（修改）现在，该函数能够锁定函数抽样时的错误
reshape	（修改）现在，语句 reshape(A,…,[ ],…)能够计算输入参数中空矩阵所需的维数大小
sortrows	（修改）现在，该函数内部将调用 MEX 函数 sortrowsc 使执行速度最快。如果该函数的输入是字符串单元数组，则将调用 MEX 函数 sortcellchar 来使执行速度最快
strfind	查找该函数中第二个字符串变量是否在第一个字符串变量中出现，以及出现的次数
svd	（修改）现在，该函数只能返回前两个输出参数，即 U 和 S
tetramesh	绘制执行 delaunayn 所需要的四面体网格图
triplot	绘制执行 delaunay 所需要的二维三角图

Matlab 6.5 R13（发布日期：2002 年 8 月）

功能	描述	出现章节
JIT 加速	具有特定属性的 For 循环现在可以以最快速度执行，这样，在某些条件下，用户就不必要采用向量化措施	11
动态结构体字段名	不再需要使用 getfield 和 setfield 函数，只需使用 var.(fstr)语法就可以寻址 var 变量中由 fstr 指定的字段名，其中 fstr 为一个字符串	8
变量和函数的最大名称长度	现在，变量名和函数名长度最长可达 63 个字符	2
整型数组下标	数组下标必须是大于零的整数值或逻辑值，例如，x(1.3)将产生错误	5
关系运算符和 64 位整型数组	（Matlab 6.5.1 版本引入）所有的关系运算符都支持 int64 和 uint64 两类数据类型	7
延迟差分方程	支持对延迟差分方程的求解	25
用于 Pentium（奔腾）4 的 BLAS（基本线性代数子函数）	提供了运行于奔腾 4 处理器上的特定的 BLAS	
支持 UMFPACK（非对称多前沿软件包）	如果需要，现在可以使用 UMFPACK 库函数求解稀疏矩阵	17
正规表达式	Matlab 现在支持正规表达式	9
逻辑类	现在，逻辑类是一个独立的 Matlab 类，该类使用一个字节保存每个数组元素	10
逻辑值 True 和 False	现在，Matlab 能够使用函数 true 和 false 直接创建由逻辑值 True 和 False 组成的逻辑数组	10
空数组比较	现在，[ ]==[ ]和[ ]==scalar 两条语句将返回空数组结果，以便与其他操作符一致	10
稀疏类	现在，稀疏是底层变量类的一个属性，而不是一个独立的变量类。在 Matlab 6.5 中，逻辑和双精度数据类型都可以是稀疏的	17
格式化错误和告警字符串	现在，函数 error 和 warning 都可以接受格式化数据输入，这一点与 sprintf 函数类似	12
消息标识符	现在，错误和告警消息都可以包含一个标识符，使得该消息具有惟一可识别性	12
告警控制	用户可以控制部分告警信息不被显示出来	12
“避绕”式逻辑操作符	使用新增的避绕式比较操作符&&和||，可以在前部分比较表达式成立时，绕过后面的比较表达式向后执行	10
定期执行	使用新增的定时器对象，用户可以按照指定的时间执行 Matlab 代码	4
文本对象属性	现在，文本对象将具有与背景框有关的属性	31

函数	描述
audiodevinfo	获取安装在 Windows 操作系统中的音频设备的信息
cdfepoch	将 Matlab 数据值或字符串转换为 CDF 格式
cdfwrite	将数据写入到 CDF 文件中
cell2mat	将矩阵单元数组组合成一个矩阵，该函数曾是神经网络工具箱（Neural Networks Toolbox）的一部分
colormapeditor	交互式颜色表编辑器
copyfile	（修改）现在，该函数也可用于拷贝路径
corrcoef	（修改）现在，该函数增加了 3 个新的语法格式
dde23	该函数将使用固定延迟量求解延迟差分方程
ddeget	从 DDE 选项结构中获取属性
ddeset	在 DDE 选项结构中创建或修改属性
deval	（修改）现在可以直接接受 dde23 的输出作为输入
false	创建由逻辑值 False 组成的数组
fileattrib	设置或获取文件的属性
gallery	（修改）现在，该库中又新增了一些测试矩阵
imformats	利用此函数可以使增加新文件类型的读写功能变得轻松自如
int64、uint64	创建有符号和无符号 64 位整型数组
isequal	（修改）当用于结构体比较时，输入参数的字段创建顺序不再要求必须一致
isequalwithequalnans	如果两个包含 NaNs 的数组被认为是相等的，就返回 True
ismember	（修改）现在，语句[tf,idx]=ismember(…)将返回指定元素的索引值 idx，并且，该语句将调用 MEX 函数 ismemc2 使执行速度最快
issorted	判断数组是否经过了排序
lasterror	返回最后一个错误消息及与之相关的信息
lu	（修改）现在，该函数可以使用 UMFPACK 算法处理稀疏矩阵
mat2cell	将一个矩阵分解为一个个矩阵单元数组，该函数曾是神经网络工具箱的一部分
movefile	（修改）现在，该函数也可用于重命名一个文件或路径
multibandread	该函数支持从原始二进制文件中读取数据
nultibandwrite	该函数支持将数据写入原始二进制文件
namelengthmax	返回最大的变量和函数名长度
orderfields	将结构体的各字段按顺序排列
perl	使用一个操作系统中的可执行文件调用 Perl 脚本
profview	创建图形化的剖析报告
psi	执行双伽玛函数（Digamma Function）
qrdelete、qrinsert	（修改）现在，这两个函数不仅可以插入和删除列，而且可以插入和删除行
regexp	寻找匹配的正规表达式

（续表）

函数	描述
regexpi	寻找匹配的正规表达式，忽略大小写
regexprep	使用正规表达式代替字符串
rethrow	重新执行错误代码
rmdir	删除一个路径，并且可以通过可选项删除该路径中的所有内容
sendmail	发送 E-mail
timer	创建和控制一个计时器对象，用于使 Matlab 代码按计划执行
triplequad	计算三重积分
true	创建由逻辑值 True 组成的数组
urlread、urlwrite	使用 URL 读写网页内容
Ver	（修改）现在，该函将返回更多更详细的内容，不过，不再返回 hostid 信息
winopen	该函数仅用于 Windows 操作系统中，用于在相应的应用程序中打开一个文件
xmlread、xmlwrite	读写 XML 文档
xslt	使用 XSLT 引擎转换 XML 文档
zip、unzip	压缩与解压缩文件和路径

Matlab 7.0 R14（发布日期：2004 年 6 月）

功能	描述	出现章节
区分大小写	现在，函数和路径名是区分大小写的。而在以前的版本中，只有在 UNIX 操作系统中才区分大小写	4，12，14
函数优先级	现在，内置函数、M 文件函数具有相同的调用优先级。而在以前的版本中，内置函数具有较高的优先级	12
非双精度的数学运算	现在，Matlab 支持基于整数数据类型和单精度数据类型的数学运算，并且，大多数内置函数现在也能够支持上述数据类型	8
匿名函数	它是写在一行中的函数声明。该函数用来取代内联函数	12
嵌套函数	现在，M 文件函数可以实现嵌套，从而允许主函数和嵌套函数共用同一个工作区	12
Unicode 字符存储	现在，Matlab 将把字符串编码为 Unicode 格式	9
Java1.4	现在，Matlab 内部使用 Java 1.4	
块注释	现在，Matlab 可以使用%{和%}语法实现块注释	4
代码单元	Matlab 编辑器允许创建和执行一部分代码，这部分代码称为代码单元	4
结果发布	现在，M 文件和图形可以被发布到 HTML、XML、LaTeX、Word 和 Powerpoint 等文件格式	
正则表达式	Matlab 7 支持扩展的正则表达式	9
自由形式的日期和时间	现在，日期和时间函数支持用户指定的日期和时间规范	16
文件 ftp 支持	现在，在 Matlab 中可以使用 ftp 命令访问文件	14

（续表）

功能	描述	出现章节
注释层	现在，图形窗口会在有注释的地方提供一个注释层。注释层包括矩形、椭圆、箭头、双向箭头、文本箭头、文本框、直线、颜色条以及图例框等	31
绘制对象	现在，高级或特殊的绘图函数将创建绘制对象，并且绘制对象具有在创建时指定的属性。过去，这些函数将只返回内核图形对象的句柄，并且内核图形对象是创建图形的主要对象	31
组对象	现在，图形对象可以被相互组合和链接	31
工具条	一种新的用户接口对象	32
面板和按钮组	一种新的用户接口容器对象	32
普通动态链接库	在 Windows 操作系统中，Matlab 支持与普通 DLL 之间的交互	36
TeX 支持	现在，用户可以通过选项设置使文本对象支持完整的 TeX 功能	31
代码检查	现在，用户可以使用 mlint 或 mlintrpt 函数访问 M 文件，并找出潜在的问题，以提高程序质量	13

数学函数	描述
acosd、acotd、acscd、asecd、asind、atand	反三角函数，返回以度数为单位的角度
balance	（修改）现在，该函数将返回不同的输出，并且不需要对数组的行和列进行重排就可以实现数据均衡
bvp4c	（修改）现在，该函数可以用于求解多点边界值问题
convhulln、delaunayn、voronoin	（修改）现在，这些函数支持用户可设置的选项
cosd、cotd、cscd、secd、sind、tand	输入参数的单位为度数的三角函数
decic	计算 ODE15I 所需的一致性初始条件
deval	（修改）现在，用户也可以通过选项设置返回指定点处的导数
eps	（修改）现在，该函数可以通过参数指定单精度或双精度，并且返回相对于任何值的 eps，而不仅仅相对于 1
expml	精确计算 exp(*x*)-1 的值
eye、ones、zeros	（修改）现在，用户可以通过这些函数的最后一个参数指定结果的数值数据类型
fftw	在 FFTW 库中调整或设置计算 FFT 所需的选项
fminbnd、fminsearch、fzero	（修改）现在，这些函数支持在每一个循环体内部调用一个外部函数
interp1、ppval、spline	（修改）现在，这些函数支持多维的数组 Y
interp1	（修改）现在，用户可以通过设置使该函数返回一个执行 ppval 所需的一个 pp 格式的参量

（续表）

数学函数	描述
linsolve	在给定结构体 A 的情况下，求 $Ax=y$ 的解
ltitr	（修改，曾是无文档函数）线性时不变时间响应内核函数
log1p	精确计算 log(1+x)的值
mimofr	（修改，曾是无文档函数）线性时不变频率响应内核函数
nthroot	求第 n 个实根
ode15i	由显式方程组成的常差分方程的解算子
odextend	常差分方程的扩展解
ordqz	对 QZ 因式分解进行重新排序
ordschur	对 Shur 因式分解进行重新排序
pwch	分段三次 Hermite 插值
quadv	向量化的 quad 函数
sort	（修改）现在，用户可以通过设置该函数的最后一个参数指定排序的方向

逻辑函数	描述
iscom	判断是否是 COM 或 ActiveX 对象
isevent	判断是否是对象的事件
isfloat	判断是否是浮点数
ishghandle	判断是否是句柄图形对象的句柄
isinteger	判断是否是整数
isinterface	判断是否是 COM 接口
ispuma	判断计算机是否运行于 Mac OS X 10.1.x
isscalar	判断是否是标量
isstrprop	判断字符串元素是否满足一系列规范
isvector	判断是否是向量（包括行向量和列向量）

M 文件处理函数	描述
auditcontents	审核指定路径中的 contents.m 文件
deleteconfirm	利用一个对话框确认文件删除操作
deprpt	扫描一个文件或路径，以查找某种相关性
diff2asv	将文件与自动保存的文件相比较
diffrpt	可视化路径浏览器
dofixrpt	扫描一个文件或路径，以查找所有的 TODO、FIXME 或 NOTE 等消息
helprpt	扫描一个文件或路径，以查找其中的帮助
makemcode	生成 M 文件，用于创建对象及其子对象

（续表）

M 文件处理函数	描述
makecontentsfile	生成一个新的 contents.m 文件
mlint、mlintrpt	检查一个 M 文件或 M 文件所在的目录，以找出潜在的问题，并提出改进的建议
path2rc	（废弃）现在已经被 savepath 代替
publish	运行一个 M 脚本文件并保存结果
recycle	确定一个被删除的文件是否被放入回收站
restoredefaultpath	恢复默认的 Matlab 路径设置
savepath	保存当前的 Matlab 路径，用于代替 path2rc 函数

图形和 GUI 函数	描述
addsubplot	在指定的位置添加一个子图形
ancestor	获取图形对象的父对象及祖先对象
annotation	添加注释对象
axes	（修改）现在，该对象新增了 ActivePositionProperty、OuterPosition 和 TightInset 三个属性
axescheck	根据输入列表处理主坐标轴对象
axis	（修改）现在，该函数第一个输入参数可以是坐标轴对象句柄
commandhistory	打开命令行历史列表窗口，或选择已打开的命令行历史列表窗口
commandwindow	打开命令行窗口，或选择已打开的命令行窗口
datacursormode	在绘制的图形上交互地创建数据光标
exportsetupdlg	显示图形导出类型选择对话框
figure	（修改）现在，该对象新增了 DockControls 属性，修改了 KeyPressFcn 属性
figureheaderdlg	显示图形头对话框
figurepalette	显示或隐藏图形的面板
getpixelposition	获取对象以像素为单位的位置向量
hasbehavior	设置或获取句柄图形对象的行为
hgexport	导出图形
hggroup	创建一个句柄图形组对象
hgtransform	创建图形变换对象
hold	（修改）现在，该函数支持 all 选项，该选项将保证在绘制过程中，后面的图形不会影响原来的颜色和线型顺序
linkaxes	使指定各坐标轴对象的范围一致
linkprop	使相应的句柄图形对象属性的值保持一致
makehgtransform	创建图形变换矩阵
pan	交互式改变绘制视角
plottools	显示或隐藏图形的绘制编辑工具栏

（续表）

图形和 GUI 函数	描述
printdlg	显示打印对话框
printpreview	显示图形的打印预览效果
propertyeditor	显示或隐藏图形的属性编辑器
refreshdata	刷新绘图中的数据
setpixelposition	设置对象以像素为单位的位置向量
showplottool	显示或隐藏图形的一个绘制编辑工具组件
title	（修改）现在，该函数第一个输入参数可以是坐标轴对象句柄
uibuttongroup	创建按钮组（buttongroup）对象
uicontainer	创建容器（container）对象
uicontrol	（修改）现在，uicontrol(h)将用户控制焦点转移到句柄为 h 的 uicontrol 对象。多行 edit 类型的 uicontrol 对象将具有一个垂直滚动轴。所有的 uicontrol 对象都具有 KeyPressFcn 回调
uigetfile	（修改）现在允许同时选择多个文件
uipanel	创建 uipanel 容器对象
uipushtool	在 uitoolbar 对象中创建按钮
uitable	创建 uitable 对象
uitoggletool	在 uitoolbar 对象中创建开关按钮
uitoolbar	创建 uitoolbar 对象
uitree	创建 uitree 对象
uitreenode	在 uitree 组件中创建节点对象
uiwait	（修改）现在，uiwait(handle,t)语句使 handle 执行后再延时时间 t，然后再执行下面的语句
xlabel、ylabel、zlabel	（修改）现在，这些函数的第一个输入参数可以是坐标轴对象句柄

外部接口函数	描述
calllib、libfunctions、libfunctionsview、libisloaded、libpointer、libstruct、loadlibrary、unloadlibrary	通用 DLL 接口函数
callsoapservice	将一个 SOAP 消息发送到一个端点
eventlisteners	列出所有的已注册事件
ftp	创建 FTP 对象
instrfindall	寻找具有指定属性值的所有串口对象
javaaddpath	在动态 Java 路径中添加目录
javaclasspath	获取和设置 Java 路径

（续表）

外部接口函数	描述
javacomponent	创建一个 Java AWT 组件，并将其放入一个图形中
javarmpath	从动态 Java 路径中删除目录
registerevent	在运行时注册指定控件的事件
unregisterallevents	在运行时注销指定控件的所有事件
unregisterevent	在运行时注销指定控件的事件

其他函数	描述
accumarray	利用累加创建数组，也就是说，如果数组中的一个元素被不止一次赋值，那么每次赋值就产生一次累加，而不是通常的覆盖
addtodate	修改一个日期数中指定的部分（如月份）
audiorecorder、audioplayer	（修改）现在，该函数既支持 UNIX 操作系统，又支持 Windows 操作系统
cast	将一个变量变换为不同的数据类型
docsearch	在 Help 浏览器中搜索 HTML 文档
find	（修改）现在，用户可以通过一个可选参数指定该函数所返回的索引值的上限，而不管函数是从头还是从尾开始搜索
fixquote	使字符串两端都具有引号（即，如果字符串中只有一个引号，则再在另一端添加一个引号）
genvarname	根据候选名称产生变量名
hd5info、hd5read、hdf5write	针对 HDF5 文件的信息显示和读写
hex2num、num2hex	将一个数字转化为 IEEE 的十六进制格式，或将一个 IEEE 十进制格式的值转化为数字
Intmax、intmin	给定整数数据类型的最大和最小整数值
intwarning	该函数用于控制整数数据类型告警的状态
mmcompinfo	多媒体压缩器的信息
mmfileinfo	获取一个多媒体文件的信息
save	（修改）现在，该命令支持对 MAT 文件的压缩
strfind、strtok	（修改）现在，这两个函数支持单元数组作为输入
strtrim	删除一个字符串中第一个非空白字符之前和最后一个非空白字符之后的空白字符
textscan	将文本文件中的内容读取到一个单元数组中。该函数比 textread 具有更多的特性
timerfindall	寻找指定属性值的计时器对象
xlswrite	将矩阵写入 Exel 电子表格中